Programación y práctica con el sinclair QL



Carlos Galán Pascual

PARANINFO S.A.

Programación y práctica con el sinclair QL

CARLOS GALAN PASCUAL

Licenciado en Informática. Profesor de la
Facultad de Informática de Madrid y del
Instituto Nacional de Administración Pública.

Programación y práctica con el sinclair QL

1985



MADRID

IMPRESO EN ESPAÑA
PRINTED IN SPAIN

ISBN: 84-283-1424-1

Depósito legal: M. 34.384.—1985



Magallanes, 25 - 28015-MADRID (3-3517)

Artes Gráficas Benzal, S. A. - Virtudes, 7 - 28010 Madrid

A Isabel.

Indice de materias

Prólogo

Cuando poco después del verano de 1984 tuve en mi poder el "QL" ya había tenido tantas referencias habladas de él que, para mí, no fue, desde luego, una máquina absolutamente nueva.

Fue entonces cuando varios de mis alumnos se sintieron atraídos por la conjunción que suponía un ordenador con estructura interna de registros de 32 bits y 128 Kbytes de memoria y el presumible buen precio con el que saldría al mercado. Entre ellos y varias otras personas que contemplaron y utilizaron mi QL me animaron a escribir un libro –el primero en lengua castellana y hecho por españoles– (1) sobre éste ordenador que poco tardaría en convertirse en uno de los más conocidos dentro del ambiente de la microinformática.

Puesto en contacto con mi editor de siempre, me instó con entusiasmo al desarrollo de la idea, y es ahora, al cabo de algunos meses cuando doy por concluida esta pequeña obra en la que he pretendido verter parte de mis experiencias con este ordenador y evitar algunos errores que he podido apreciar en otras ediciones extranjeras, en las que por excesiva premura de edición, solamente han tocado aspectos superficiales de la máquina y del lenguaje SuperBASIC.

No obstante, existirán sin duda errores no intencionados en estas páginas, que espero que mis lectores sepan disculpar con su proverbial comprensión.

Este libro está concebido fundamentalmente como una obra didáctica destinada a todos aquellos que posean o utilicen el Sinclair QL y aún más, a todos aquellos que deseen introducirse en el terreno de la programación estructurada utilizando un lenguaje de alto nivel ya bastante evolucionado como es el SuperBASIC.

---

(1) N. del E.: En realidad mientras se ha preparado la edición de esta obra ha aparecido algún texto sobre este tema.

No se ha presupuesto a priori una imagen de alumno "ideal", puesto que la información contenida en este volumen es válida tanto para los no iniciados sin conocimientos de informática como para los que ya poseen estas enseñanzas de BASIC y de microordenadores o incluso tienen o han utilizado otra marca o modelo de ordenador.

Para aquellos que se encuentren en el primer grupo, esto es, para los que este libro es su primer contacto con éste mundo, se ha escrito el capítulo 1 dedicado a mostrar algunos conceptos previos sobre informática básica. Todos aquéllos que ya posean estos conocimientos primarios pueden muy bien omitir la lectura de este capítulo.

El capítulo 0 contiene un primer acercamiento a la máquina propiamente dicha respecto de su hardware, software y paquetes de aplicación.

En el capítulo 2 se inicia al lector en la programación en Super BASIC y se exponen las características básicas de éste lenguaje y las primeras sentencias de programación.

Las instrucciones de Entrada/Salida se estudian con detalle en el capítulo 3.

Las estructuras de datos de matrices y cadenas de caracteres son estudiadas profusamente en el capítulo 4, así como las instrucciones y funciones para su manejo.

En el capítulo 5 se introduce al lector en las instrucciones de bifurcación y sentencias alternativas, así como las diferentes modalidades de relaciones que pueden imponerse dentro del contexto de un programa.

Las instrucciones repetitivas de programación estructurada se contemplan con detalle en el capítulo 6, ya que conforman una buena parte de la potencia de proceso del SuperBASIC.

En los capítulos 7 y 8 se inicia al lector en el terreno de la subprogramación. Primero se estudian las instrucciones para subprogramación clásica y después se trata con detalle los conceptos de funciones y procedimientos que hacen a este lenguaje una poderosa herramienta.

Los gráficos y las instrucciones necesarias para crearlos y tratarlos son estudiadas profusamente en el capítulo 9 donde se descubre al lector las inmensas posibilidades en este sentido que encierra la máquina.

El capítulo 10 está íntegramente dedicado a la declaración y manejo de ficheros de datos como soporte fundamental para el almacenamiento y recuperación de grandes cantidades de informaciones homogéneas.

El capítulo 11 sirve de complemento a todo lo estudiado con anterioridad y en él se vierten algunos conceptos importantes para entender el color, el reloj del sistema, los mensajes de error, la estructura interna de la memoria y del Sistema Operativo QDOS, las redes de comunicaciones, el sonido y varias otras características fundamentales.

En el Apéndice A se muestran y detallan todos los comandos de uso del SuperBASIC y del QDOS y el lector deberá ir consultando éste apartado a medida que avanza en el estudio del texto.

El Apéndice B muestra todo el repertorio de las palabras reservadas del SuperBASIC y del QDOS con una traducción de su función específica.

El texto se completa con una bibliografía sobre conceptos vertidos en los capítulos anteriores y un Indice Alfabético de Conceptos del texto.

0

Introducción al QL

El más potente de los microordenadores de Sinclair Research Limited fue presentado oficialmente en Londres el día 12 de Enero de 1984, y fue bautizado con el nombre de *"QUANTUM LEAP"* (Salto Cuántico). Todo el mundo empezó a llamarle *QL*.

El Sinclair QL pretende introducirse dentro del terreno profesional de los microordenadores y para ello ha sido dotado de las características hardware y software adecuadas, como apuntaremos en éste mismo capítulo preliminar y que profundizaremos en los restantes.

Seguidamente daremos un breve repaso técnico a las características más sobresalientes de este ordenador en lo referente a su hardware, software y paquetes de aplicación. Aunque lo mejor, amigo lector, es sin duda, sentarse delante de esta máquina y observar detenidamente todas las posibilidades que ofrece y que estudiaremos a lo largo de este libro.

0.1. EL HARDWARE

El QL se fundamenta en el diseño monobloque tradicional de Sinclair, donde se encuentran disponibles: el teclado, toda la circuitería, los slots para microdrives y las puertas de entrada/salida para las conexiones exteriores. El único elemento que se ha mantenido fuera de esta carcasa ha sido la fuente de alimentación, cuyo volumen y calor disipado hacían inviable su inclusión dentro del conjunto anterior.

El teclado es del tipo "QWERTY" de aspecto y tacto profesional y posee, amén de todas las teclas del alfabeto, con posibilidad de mayúsculas y minúsculas, cinco teclas de función para usos específicos, especialmente diseñadas para su uso conjunto con los cuatro paquetes de aplicaciones estandar que acompañan a la máquina.

Asimismo, dispone de todo un juego de teclas con caracteres especiales que cubren completamente el espectro de necesidades de escritura y representación. Dispone también de las cuatro tradicionales teclas con flechas para el manejo del cursor por todo lo largo y ancho de la pantalla de visualización.

Internamente, el QL funciona gracias a la potente ayuda prestada por dos microprocesadores. El primero de ellos es el 68008 de Motorola que ejerce las funciones de Unidad Central de Proceso y que posee una arquitectura interna de registros (como se verá con detalle en el Apéndice B) de 32-bits y una frecuencia de reloj de 7,5 MHz, todo ello con capacidad para direccionar 1 Mbyte de memoria. El juego de instrucciones de este procesador es muy amplio, de forma que se cubren todas las posibles necesidades de trabajo con lenguajes de alto nivel de tipo estructurado. El Apéndice B muestra el repertorio completo de tales instrucciones.

Posee, además, un microprocesador adicional, el 8049 de Intel que es el encargado de controlar las entradas por teclado, la transmisión de datos a través de las puertas estandar RS-232C y el sonido. Una descripción más detallada de este microprocesador también puede encontrarse en el Apéndice B.

Dispone también de varios circuitos integrados de diseño específico para esta máquina y que se encargan de controlar la pantalla, la memoria, la red de área local QLAN y las funciones de los microdrives para almacenamiento y recuperación de ficheros y programas.

En la versión estandar, el QL está equipado de origen con una memoria RAM de 128 Kbytes, susceptible de ser ampliada mediante módulos conectables directamente de 512 Kbytes. Posee igualmente una memoria ROM de 32 Kbytes que contiene el Sistema Operativo QDOS y el intérprete del lenguaje SuperBASIC. Parte del capítulo 11 está dedicado a estudiar con detalle las características del QDOS. Una conexión de expansión prevista en el aparato, hace que sea posible conectarle cartuchos adicionales de ROM con el objeto de incorporarle compiladores de lenguajes, sistemas operativos diferentes, depuradores, etc.

El QL puede ser conectado tanto a un monitor en color o monocromo como a un televisor de uso doméstico, de forma que puede trabajarse indistintamente con dos tipos posibles de resolución. El capítulo 9 dedicado a los gráficos trata este tema con profundidad.

En el modo de "alta resolución" pueden representarse gráficos sobre una pantalla de 512 x 256 pixels sobre cuatro colores base: blanco, negro, rojo y verde. En el modo de "resolución media" pueden representarse gráficos de 256 x 256 pixels sobre ocho posibles colores base.

Respecto del sonido, el QL dispone de varias instrucciones para el manejo de posibilidades sonoras de forma que se tienen en cuenta muchos de los parámetros que conforman una señal audible y que son susceptibles de ser controlados y programados de manera que se posee un infinito campo de investigación y experimentación en este sentido.

El reloj de tiempo real permite tener acceso inmediato a la fecha y a la hora concreta y posee varios comandos e instrucciones para su manejo y actualización. Ambas características: sonido y reloj son tratadas con detalle en el capítulo 11.

El QL dispone de dos entradas incorporadas para memoria auxiliar y que son los llamados *microdrives*, que constan de un cartucho de banda magnética o cinta sin fin que permiten un acceso aleatorio de los datos y programas con un tiempo medio de acceso de 3,5 segundos. El número de microdrives conectables puede ascender a 6 unidades, aunque puede conectarse un interface para discos flexibles o incluso Winchester con una capacidad de almacenamiento de 5 ó 10 Mbytes.

La capacidad mínima de cada microdrive es de 100 Kbytes, siendo su velocidad de transferencia de información de 15 Kbytes por segundo.

El capítulo 11 trata con detalle el manejo y uso de éstos microdrives.

En la parte posterior de la carcasa pueden observarse dos puertas seriales tipo RS-232-C utilizables para conectar periféricos con este interface (impresoras, plotters, modems, etc) y que pueden transferir información a velocidades comprendidas entre 75 y 19.200 baudios. Pudiéndose conectar un interface paralelo tipo Centronics opcionalmente.

Sin olvidar del todo los juegos, el QL ha previsto dos puertas de conexión para dos joysticks para su utilización con programas de ocio o incluso de aprendizaje.

Se incluye también en el aparato un interface para una red de área local QLAN capaz de comunicar entre sí hasta 64 unidades QL o Spectrum con una velocidad de transmisión de 100 Kbaudios, de forma que varias unidades puedan estar accediendo al mismo (o distintos) ficheros y/o compartiendo los mismos recursos, tanto software como hardware. En el Capítulo 11 se estudia esta configuración de red local.

0.2. EL SOFTWARE

El Sistema Operativo del QL es *multitarea*, lo que permite la ejecución simultánea de varios programas. Una posible forma de realizar esto, consiste en dividir la pantalla de visualización en *ventanas* (windows) donde cada una de ellas refleja la situación actual de un programa determinado. Una descripción detallada de las posibilidades del Sistema Operativo QDOS puede encontrarse en el capítulo 11.

Respecto del lenguaje de programación *SuperBASIC* contenido en ROM, debemos decir que se trata de una versión muy avanzada respecto del tradicional BASIC y que, aún siendo compatible en forma ascendente, dispone de potentes instrucciones de *programación estructurada, funciones* y *procedimientos* con parámetros, que lo hacen perfectamente equiparable a segmentos de otros lenguajes de programación como C o PASCAL.

Un buen ejemplo de esto lo constituye la declaración y utilización de procedimientos *(procedures)* que independizan las tareas aisladas de un programa, como veremos con profundidad en el capítulo 8.

La mayor parte de este libro está dedicado a examinar con detalle, añadiendo gran cantidad de ejemplos y ejercicios resueltos, este lenguaje SuperBASIC que ha supuesto un paso de gigante en el desarrollo y la implementación de programas estructurados con el uso de intérpretes.

0.3. PAQUETES DE APLICACION

Junto con el ordenador, se incluyen cuatro paquetes desarrollados por la firma PSION Limited y que constituyen unas herramientas habituales para el tratamiento de la información.

El primero de ellos es el denominado *QL QUILL* y se trata de un macroprograma para *tratamiento de textos*, utilizando un sistema paramétrico, de forma que siempre se tiene en pantalla toda la información necesaria para crear, modificar, salvar, imprimir, etc., el documento. Siempre puede utilizarse la tecla de función F1 como ayuda (HELP) para lo que se pretenda hacer, descubriéndose una completísima documentación de todas las posibilidades.

En este paquete están contempladas todas las posibilidades requeridas a un buen procesador de textos y que conforman uno de los programas más importantes para su uso con microordenadores.

El *QL ABACUS* es en esencia una *"hoja electrónica"* para tabulación, planificación, cálculo y almacenamiento de la información de forma que ésta puede estar dividida en celdas de 255 filas por 64 columnas, cada una de las cuales puede contener informaciones de tipo numérico, alfanumérico o incluso textos; pudiendo ser calculadas unas a partir de otras introduciendo las fórmulas matemáticas adecuadas.

El *QL ARCHIVE* es un programa para creación, actualización y recuperación de informaciones en una *base de datos*. La forma de creación de un archivo, según éste programa, es absolutamente visual de manera que el usuario sabe en todo momento cuál es la estructura de los datos, su contenido y el número de los mismos. Dispone asimismo de una variadísima gama de comandos que permiten la clasificación (sorting) de ficheros, la búsqueda y selección de aquellos registros que cumplan determinadas condiciones (searching) y la edición de informes específicos (reporting) de forma que no es necesario indicar a priori cuán largo será un registro ya que admite registros de longitud y campos variables.

Y por último, el *QL EASEL* es un macro-programa para la creación de gráficos comerciales de todo tipo, desde histogramas a diagramas de Gantt, pasando por gráficos con secciones circulares, etc. Se trata, como todos los anteriores, de un programa interactivo de forma que el usuario no tiene porqué recordar los nombres de los comandos a utilizar, ya que es el propio programa quien le va guiando según sus necesidades particulares en cada caso.

Una importante ventaja de estos cuatro paquetes es que está permitido el intercambio de informaciones y de datos entre ellos de forma que, por ejemplo, el QL QUILL (procesador de textos) puede requerir la presencia en un instante determinado de un gráfico generado por el QL EASEL e incluirlo dentro del propio documento que se esté tratando.

Como hemos dicho, estos cuatro programas de aplicación se incluyen de origen con el ordenador. No obstante, PSION Limited y varias firmas más de construcción de software disponen de múltiples aplicaciones susceptibles de ser ejecutadas con el QL y que sin duda se verán incrementadas en número y calidad a lo largo del tiempo.

1

Conceptos previos sobre Informática básica

1.1. INTRODUCCION

Dentro de este capítulo se verterán algunos conceptos introductorios necesarios para empezar a programar en el SuperBASIC del QL. Se definirán algunos aspectos de uso común y algunos conocimientos de programación en general, para abordar con posterioridad el estudio detallado de las características del lenguaje SuperBASIC que lo hacen, como veremos, muy superior a otro tipo de lenguajes BASIC's anteriores.

Aquel lector que conozca con amplitud los conceptos de programa, ordenador, códigos, lenguajes de programación, etc., puede muy bien omitir la lectura de este primer capítulo.

1.2. EL CONCEPTO DE PROGRAMA

Todas las actividades desarrolladas por una máquina no son fruto de la casualidad. Para que un ordenador efectúe un conjunto de procesos es necesario que algo le indique con *claridad* y *precisión* que es lo que tiene que hacer. Este algo de lo que hablamos es lo que se entiende por *programa*.

Un programa es un conjunto de órdenes que la máquina obedece durante la ejecución del mismo. Obvio resulta mencionar que tal conjunto de órdenes debe estar dispuesto de forma que sean comprensible por el ordenador de que se trate y por consiguiente, estar sujeto a unos códigos especiales que la máquina debe entender.

Cuando una misma máquina puede realizar diferentes acciones en base a la ejecución de otros tantos programas diferentes, entonces se dice que la máquina es *programable.*

La posibilidad de que en el conjunto de órdenes dadas a la máquina (instrucciones, sentencias o comandos) se encuentre alguna con capacidad para decidir en un instante del programa, la acción que se ejecutará seguidamente es sin duda lo que diferencia a los ordenadores de las máquinas de cálculo en general.

En los siguientes capítulos estudiaremos con detalle estas instrucciones de decisión que, junto con las restantes, conforman el repertorio de órdenes que podemos utilizar al escribir un programa en Super BASIC.

1.3. EL CONCEPTO DE ORDENADOR

El ordenador, en nuestro caso, el Sinclair QL, es la máquina que se utiliza para ejecutar los programas previamente escritos con el objeto de obtener unos determinados resultados.

La figura 1-1 muestra un esquema de los componentes de un ordenador en general.

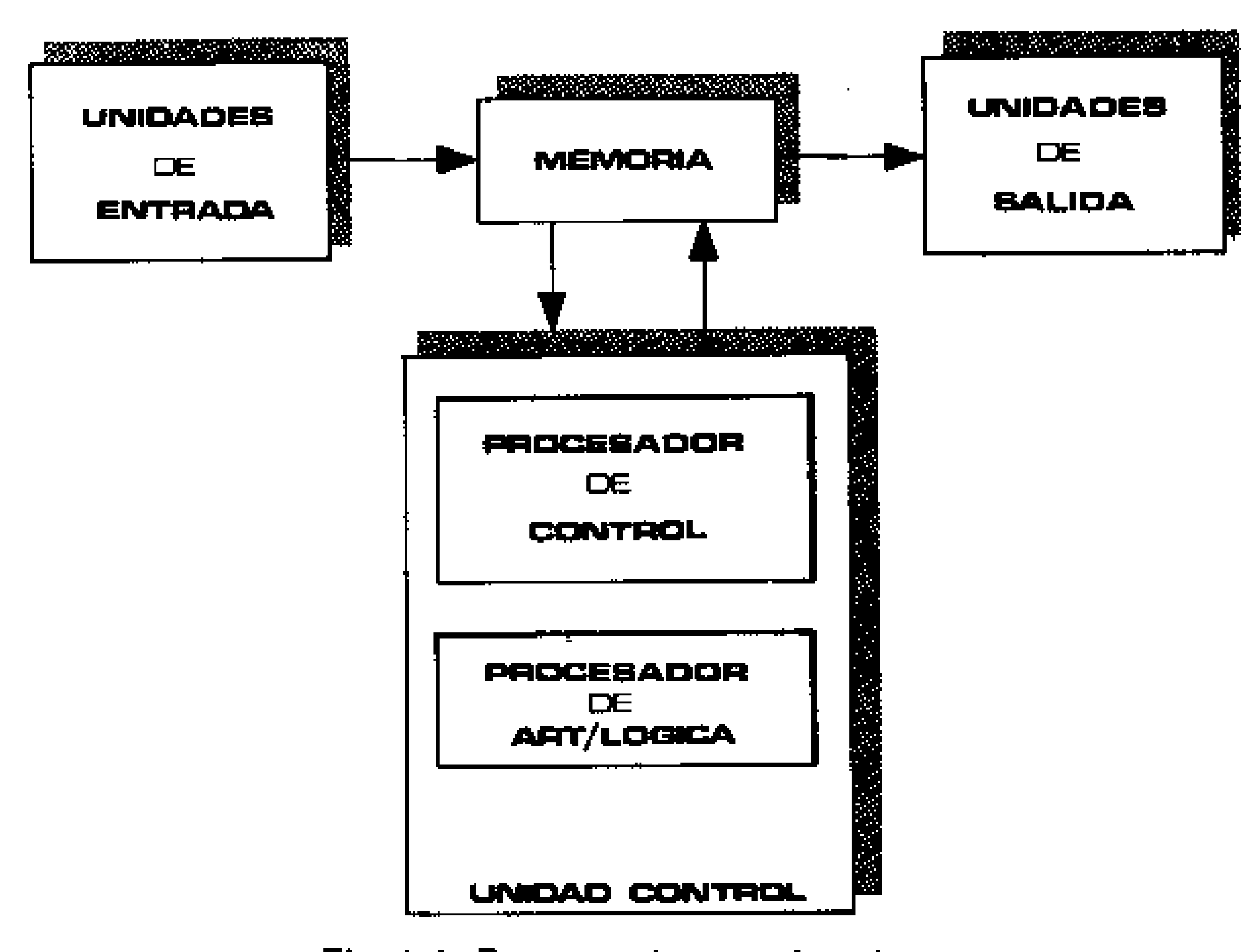


Fig. 1.1. Esquema de un ordenador.

Las *UNIDADES DE ENTRADA* se utilizan para introducir datos al sistema. Son, en definitiva, el puente entre el mundo exterior y el interior de la máquina. En nuestro caso, las unidades de entrada más frecuentes serán: el propio teclado del QL mediante el cual introduciremos datos y programas al ordenador, los microdrives cuyo contenido pueden ser también datos y/o programas y cualquier periférico conectado vía red local QLAN (tales como otros ordenadores enviando datos) o incluso líneas telefónicas, los joysticks, etc.

El capítulo 11 de este libro trata con detalle todos estos elementos periféricos conectables.

Las *UNIDADES DE SALIDA* son utilizadas para informar sobre los resultados obtenidos mediante un proceso controlado bajo programa. Pueden servir para este cometido: la pantalla o monitor conectado a nuestro ordenador, los microdrives receptores de información de salida, las puertas de red local para enviar información a otras estaciones de trabajo y por supuesto las impresoras, dispositivos muy conocidos, donde la información resulta impresa en papel mediante una cadena, barra, bola, margarita o matriz de agujas que conforman los caracteres habituales del alfabeto más los signos de cálculo gráficos, de puntuación o especiales.

La *MEMORIA* o *ALMACENAMIENTO PRINCIPAL (RAM)* es una unidad utilizada para almacenar las instrucciones y/o los datos manejados por un programa mientras esté siendo ejecutado. De la rapidez de acceso y de su capacidad dependen en gran medida las posibilidades de proceso del ordenador en cuestión.

La información se almacena en la memoria en bits. A la agrupación de estos bits en unidades de acceso le denominaremos *byte* u *octeto* (8 bits), *palabra* (16 bits) y *palabra-larga* (32 bits). Como aspecto importante es conveniente hacer notar que para que una instrucción sea ejecutada debe residir en memoria en el mismo momento de su ejecución.

En nuestro caso, como ya apuntábamos en el capítulo 0, esta memoria RAM posee de origen una capacidad de 128 Kbytes.

A la *UNIDAD CENTRAL DE PROCESO (CPU)* o *PROCESADOR CENTRAL* se la considera, con toda razón, la fracción más importante o definitoria del sistema u ordenador. Se encuentra dividida en dos componentes:

a) La *UNIDAD ARITMETICA LOGICA*, encargada de realizar las instrucciones aritméticas y comparaciones lógicas del programa, y

b) La *UNIDAD DE CONTROL*, que realiza la función de coordinar el resto del sistema, es decir, los dispositivos de Entrada/Salida, las sucesivas instrucciones a ejecutar de un programa residente en memoria, mantener el diálogo con el usuario, etc.

En el Sinclair QL estas funciones las comparten los microprocesadores 68008 de Motorola y el 8049 de Intel.

1.4. LA CODIFICACION DE LA INFORMACION

A la forma de representación material de cualquier información (ya sean instrucciones de programas o datos) se denomina *código*.

En la transmisión de conocimientos por los hombres, se utilizan habitualmente dos tipos de codificaciones: la *codificación fonética* (para los mensajes hablados) y la *codificación ortográfica* (para los mensajes escritos).

El ordenador también posee su propio y exclusivo tipo de codificación.

Los símbolos básicos de la representación interna de la información en un ordenador es la *codificación binaria* (que consta, como es sabido, de dos únicos símbolos, el cero (0) y el uno (1)).

Internamente, todas las informaciones manejadas por el ordenador estarán dispuestas en forma de cadenas o tiras de ceros y de unos. Es el único lenguaje que entiende la máquina.

Una notación binaria también puede codificarse utilizando sistemas parejos como el *octal, hexadecimal,* etc. La figura 2-1 del siguiente capítulo muestra la codificación decimal y hexadecimal para todos los símbolos de representación de caracteres en *el Sinclair QL.*

1.5. LOS LENGUAJES DE PROGRAMACION

Para evitar la evidente incomodidad de tener que escribir los programas en binario se han creado los lenguajes de programación que se describen de una forma que se acerca más al lenguaje habitual de los hombres, tal es el caso del avanzado *SuperBASIC* del QL.

De esta forma, un usuario que desee confeccionar un programa, no tiene porqué saber nada acerca de la estructura interna de la máquina, sino más bien limitarse a escribir su programa de acuerdo con las reglas sintácticas del lenguaje que está utilizando.

En este libro utilizaremos y estudiaremos con detalle el SuperBASIC implementado de origen en ROM en el ordenador Sinclair QL.

Para que el QL "entienda" nuestro programa escrito en SuperBASIC es necesario que se *traduzca* a código máquina, esto es; que se transforme en listas de código binario que sean directamente interpretables por el procesador MC68008. Esta operación la realiza el *intérprete* de SuperBASIC que analiza paso a paso el programa fuente escrito por el usuario al mismo tiempo que ejecuta cada una de las instrucciones que lo conforma.

Los intérpretes permiten operar con el lenguaje fuente en forma *interactiva* realizando las modificaciones que se crean oportunas en cada momento y procediendo inmediatamente después a su ejecución.

Fig. 1.2.— Estructura del intérprete y de un programa SuperBASIC.

El SuperBASIC del QL es una versión muy avanzada de los lenguajes BASIC's tradicionales y que incorpora algunas características que lo hacen mucho más potente que sus predecesores.

Opera como un intérprete interactivo que permite la edición del programa y su posterior ejecución de forma inmediata.

La figura 1-2 muestra un esquema de tales procesos de edición y de ejecución.

2

Elementos básicos de programación

2.1. INTRODUCCION

En este capítulo se estudiarán con detalle todos aquellos elementos básicos para confeccionar programas y que son: los *juegos de caracteres* empleados para escribir los nombres utilizados en los programas, los conceptos de *constantes aritméticas* y de *cadena* (string), los *identificadores* o nombres que proporciona el usuario para mencionar a las variables de su programa, los *operadores* que relacionan a identificadores y constantes, el concepto de *expresión* como compendio de todo lo anterior, ya sea de tipo algebraico o de cadenas y se estudian las primeras instrucciones SuperBASIC como son la *sentencia de asignación* y *de comentarios.*

Se detalla igualmente una de las características más importantes de este lenguaje como es la *compatibilidad o conversión (coerción)* que lo ayudan a estar considerado como un potente lenguaje de programación.

Se describen seguidamente dos nuevas instrucciones SuperBASIC para la lectura de datos en un programa como son las sentencias READ y DATA y la instrucción RESTORE.

Con todo ello ya se dispone de los elementos básicos para empezar a comprender todo el campo siguiente que conforma el repertorio de todas las instrucciones y características del SuperBASIC.

2.2. EL JUEGO DE CARACTERES DEL QL

La tabla de la figura 2-1 muestra el juego completo de caracteres disponibles en el QL.

Decimal	Hex	Teclas	Visualización/función
0	00	CTRL £	NULL
1	01	CTRL A	
2	02	CTRL B	
3	03	CTRL C	cambio del canal de entrada
4	04	CTRL D	
5	05	CTRL E	
6	06	CTRL F	
7	07	CTRL G	
8	08	CTRL H	
9	09	TAB (CTRL I)	Campo siguiente
10	0A	ENTER (CTRL J)	Nueva línea/entrada de comando
11	0B	CTRL K	
12	0C	CTRL L	
13	0D	CTRL M	Enter
14	0E	CTRL N	
15	0F	CTRL O	
16	10	CTRL P	
17	11	CTRL Q	
18	12	CTRL R	
19	13	CTRL S	
20	14	CTRL T	
21	15	CTRL U	
22	16	CTRL V	
23	17	CTRL W	
24	18	CTRL X	
25	19	CTRL Y	
26	1A	CTRL Z	
27	1B	ESC (CTRL SHIFT [)	Cancelación del nivel actual del
28	1C	CTRL SHIFT \	comando.
29	1D	CTRL SHIFT]	
30	1E	CTRL SHIFT £	
31	1F	CTRL SHIFT ESC	
32	20	Space	Espacio
33	21	SHIFT 1	!
34	22	SHIFT '	"
35	23	SHIFT 3	#
36	24	SHIFT 4	$
37	25	SHIFT 5	%
38	26	SHIFT 6	&
39	27	'	'
40	28	SHIFT 9	(
41	29	SHIFT 0	)
42	2A	SHIFT 8	*
43	2B	SHIFT =	+
44	2C	,	,
45	2D	-	-
46	2E	.	.

Decimal	Hex	Teclas	Visualización/función
47	2F	/	/
48	30	0	0
49	31	1	1
50	32	2	2
51	33	3	3
52	34	4	4
53	35	5	5
54	36	6	6
55	37	7	7
56	38	8	8
57	39	9	9
58	3A	SHIFT ;	:
59	3B	;	;
60	3C	SHIFT ,	<
61	3D	=	=
62	3E	SHIFT .	>
63	3F	SHIFT /	?
64	40	SHIFT 2	@
65	41	SHIFT A	A
66	42	SHIFT B	B
67	43	SHIFT C	C
68	44	SHIFT D	D
69	45	SHIFT E	E
70	46	SHIFT F	F
71	47	SHIFT G	G
72	48	SHIFT H	H
73	49	SHIFT I	I
74	4A	SHIFT J	J
75	45	SHIFT K	K
76	4C	SHIFT L	L
77	4D	SHIFT M	M
78	4E	SHIFT N	N
79	4F	SHIFT O	O
80	50	SHIFT P	P
81	51	SHIFT Q	Q
82	52	SHIFT R	R
83	53	SHIFT S	S
84	54	SHIFT T	T
85	55	SHIFT U	U
86	56	SHIFT V	V
87	57	SHIFT W	W
88	58	SHIFT X	X
89	59	SHIFT Y	Y
90	5A	SHIFT Z	Z
91	5B	[	[

<table>
<tr><th>Decimal</th><th>Hex</th><th>Teclas</th><th>Visualización/función</th></tr>
<tr><td>92</td><td>5C</td><td>\</td><td>\</td></tr>
<tr><td>93</td><td>5D</td><td>]</td><td>]</td></tr>
<tr><td>94</td><td>5E</td><td>SHIFT 6</td><td>.</td></tr>
<tr><td>95</td><td>5F</td><td>SHIFT -</td><td>_</td></tr>
<tr><td>96</td><td>60</td><td>£</td><td>£</td></tr>
<tr><td>97</td><td>61</td><td>A</td><td>a</td></tr>
<tr><td>98</td><td>62</td><td>B</td><td>b</td></tr>
<tr><td>99</td><td>63</td><td>C</td><td>c</td></tr>
<tr><td>100</td><td>64</td><td>D</td><td>d</td></tr>
<tr><td>101</td><td>65</td><td>E</td><td>e</td></tr>
<tr><td>102</td><td>66</td><td>F</td><td>f</td></tr>
<tr><td>103</td><td>67</td><td>G</td><td>g</td></tr>
<tr><td>104</td><td>68</td><td>H</td><td>h</td></tr>
<tr><td>105</td><td>69</td><td>I</td><td>i</td></tr>
<tr><td>106</td><td>6A</td><td>J</td><td>j</td></tr>
<tr><td>107</td><td>6B</td><td>K</td><td>k</td></tr>
<tr><td>108</td><td>6C</td><td>L</td><td>l</td></tr>
<tr><td>109</td><td>6D</td><td>M</td><td>m</td></tr>
<tr><td>110</td><td>6E</td><td>N</td><td>n</td></tr>
<tr><td>111</td><td>6F</td><td>O</td><td>o</td></tr>
<tr><td>112</td><td>70</td><td>P</td><td>p</td></tr>
<tr><td>113</td><td>71</td><td>Q</td><td>q</td></tr>
<tr><td>114</td><td>72</td><td>R</td><td>r</td></tr>
<tr><td>115</td><td>73</td><td>S</td><td>s</td></tr>
<tr><td>116</td><td>74</td><td>T</td><td>t</td></tr>
<tr><td>117</td><td>75</td><td>U</td><td>u</td></tr>
<tr><td>118</td><td>76</td><td>V</td><td>v</td></tr>
<tr><td>119</td><td>77</td><td>W</td><td>w</td></tr>
<tr><td>120</td><td>78</td><td>X</td><td>x</td></tr>
<tr><td>121</td><td>79</td><td>Y</td><td>y</td></tr>
<tr><td>122</td><td>7A</td><td>Z</td><td>z</td></tr>
<tr><td>123</td><td>7B</td><td>SHIFT [</td><td>{</td></tr>
<tr><td>124</td><td>7C</td><td>SHIFT \</td><td>|</td></tr>
<tr><td>125</td><td>7D</td><td>SHIFT]</td><td>}</td></tr>
<tr><td>126</td><td>7E</td><td>SHIFT £</td><td>‾</td></tr>
<tr><td>127</td><td>7F</td><td>SHIFT ESC</td><td>©</td></tr>
<tr><td>128</td><td>80</td><td>CTRL ESC</td><td>ä</td></tr>
<tr><td>129</td><td>81</td><td>CTRL SHIFT 1</td><td>ã</td></tr>
<tr><td>130</td><td>82</td><td>CTRL SHIFT '</td><td>å</td></tr>
<tr><td>131</td><td>83</td><td>CTRL SHIFT 3</td><td>é</td></tr>
<tr><td>132</td><td>84</td><td>CTRL SHIFT 4</td><td>ö</td></tr>
<tr><td>133</td><td>85</td><td>CTRL SHIFT 5</td><td>õ</td></tr>
<tr><td>134</td><td>86</td><td>CTRL SHIFT 7</td><td>ø</td></tr>
<tr><td>135</td><td>87</td><td>CTRL</td><td>ü</td></tr>
<tr><td>136</td><td>88</td><td>CTRL SHIFT 9</td><td>ç</td></tr>
<tr><td>137</td><td>89</td><td>CTRL SHIFT 0</td><td>ñ</td></tr>
</table>

Decimal	Hex	Teclas	Visualización/función
138	8A	CTRL SHIFT 8	æ
139	8B	CTRL SHIFT =	œ
140	8C	CTRL ,	á
141	8D	CTRL -	à
142	8E	CTRL .	â
143	8F	CTRL /	ë
144	90	CTRL 0	ê
145	91	CTRL 1	ë
146	92	CTRL 2	ï
147	93	CTRL 3	í
148	94	CTRL 4	ı
149	95	CTRL 5	î
150	96	CTRL 6	ó
151	97	CTRL 7	ò
152	98	CTRL 8	ô
153	99	CTRL 9	ú
154	9A	CTRL SHIFT ;	ù
155	9B	CTRL ;	û
156	9C	CTRL SHIFT ,	ß
157	9D	CTRL =	¢
158	9E	CTRL SHIFT .	¥
159	9F	CTRL SHIFT /	`
160	A0	CTRL SHIFT 2	Ä
161	A1	CTRL SHIFT A	Ã
162	A2	CTRL SHIFT B	Å
163	A3	CTRL SHIFT C	É
164	A4	CTRL SHIFT D	Ö
165	A5	CTRL SHIFT E	Õ
166	A6	CTRL SHIFT F	Ø
167	A7	CTRL SHIFT G	Ü
168	A8	CTRL SHIFT H	Ç
169	A9	CTRL SHIFT I	Ñ
170	AA	CTRL SHIFT J	Æ
171	AB	CTRL SHIFT K	alpha Œ
172	AC	CTRL SHIFT L	delta α
173	AD	CTRL SHIFT M	theta δ
174	AE	CTRL SHIFT N	lambda θ
175	AF	CTRL SHIFT O	mu λ
176	B0	CTRL SHIFT P	pi μ
177	B1	CTRL SHIFT Q	phi π
178	B2	CTRL SHIFT R	psi ϕ
179	B3	CTRL SHIFT S	¡
180	B4	CTRL SHIFT T	¿
181	B5	CTRL SHIFT U	?
182	B6	CTRL SHIFT V	§
183	B7	CTRL SHIFT W	¤
184	B8	CTRL SHIFT X	<<

Decimal	Hex	Teclas	Visualización/función
185	B9	CTRL SHIFT Y	>>
186	BA	CTRL SHIFT Z	°
187	BB	CTRL [	÷
188	BC	CTRL \	←
189	BD	CTRL]	→
190	BE	CTRL SHIFT 6	↑
191	BF	CTRL SHIFT -	↓
192	C0	Left	Cursor un carácter a la izquierda
193	C1	ALT Left	Cursor al comienzo de línea.
194	C2	CTRL Left	Borrar un carácter a la izquierda.
195	C3	CTRL ALT Left	Borrar línea.
196	C4	SHIFT Left	Cursor a la izquierda una palabra
197	C5	SHIFT ALT Left	Desplazamiento a la izquierda.
198	C6	SHIFT CTRL Left	Borrar una palabra a la izquierda.
199	C7	SHIFT CTRL A Left	
200	C8	Right	Cursor un carácter a la derecha.
201	C9	ALT Right	Cursor al final de la línea.
202	CA	CTRL Right	Borra el carácter del cursor.
203	CB	CTRL ALT Right	Borra hasta el final de la línea.
204	CC	SHIFT Right	Cursor a la derecha una palabra.
205	CD	SHIFT ALT Right	Desplazamiento a la derecha.
206	CE	SHIFT CTRL Right	Borra el carácter de la derecha y del cursor
207	CF	SHIFT CTRL ALT Right	
208	D0	Up	Cursor arriba.
209	D1	ALT Up	Desplazamiento arriba.
210	D2	CTRL Up	Búsqueda hacia atrás.
211	D3	ALT CTRL Up	
212	D4	SHIFT Up	Punto superior de la pantalla.
213	D5	SHIFT ALT Up	
214	D6	SHIFT CTRL Up	
215	D7	SHIFT CTRL ALT Up	
216	D8	Down	Cursor abajo.
217	D9	ALT Down	Desplazamiento abajo.
218	DA	CTRL Down	Búsqueda hacia adelante.
219	DB	ALT CTRL Down	
220	DC	SHIFT Down	Punto inferior de la pantalla.
221	DD	SHIFT ALT Down	
222	DE	SHIFT CTRL Down	
223	DF	SHIFT CTRL ALT Down	
224	E0	CAPSLOCK	Toggle CAPSLOCK function
225	E1	ALT CAPSLOCK	
226	E2	CTRL CAPSLOCK	
227	E3	ALT CTRL CAPSLOCK	
228	E4	SHIFT CAPSLOCK	
229	E5	SHIFT ALT CAPSLOCK	
230	E6	SHIFT CTRL CAPSLOCK	
231	E7	SHIFT CTRL ALT CAPSLOCK	

Decimal	Hex	Teclas	Visualización/función
232	E8	F1	
233	E9	CTRL F1	
234	EA	SHIFT F1	
235	EB	CTRL SHIFT F1	
236	EC	F2	
237	ED	CTRL F2	
238	EE	SHIFT F2	
239	EF	CTRL SHIFT F2	
240	F0	F3	
241	F1	CTRL F3	
242	F2	SHIFT F3	
243	F3	CTRL SHIFT F3	
244	F4	F4	
245	F5	CTRL F4	
246	F6	SHIFT F4	
247	F7	CTRL SHIFT F4	
248	F8	F5	
249	F9	CTRL F5	
250	FA	SHIFT F5	
251	FB	CTRL SHIFT F5	
252	FC	SHIFT space	"Special" space
253	FD	SHIFT TAB	Back tab (CTRL ignored)
254	FE	SHIFT ENTER	"Special" newline (CTRL ignored)
255	FF	See below	

Fig. 2.1.— Juego de caracteres del QL.

Se indica el valor decimal, hexadecimal, las teclas a pulsar y el resultado obtenido para cada una de ellas.

Como ya veremos en los restantes capítulos y apartados de este libro, utilizaremos estos caracteres para representar nombres de usuario, operaciones, condiciones y, en fin, todo aquello que conforma la potente sintaxis del lenguaje SuperBASIC.

2.3. LAS CONSTANTES

Existen en SuperBASIC dos tipos de constantes: las *constantes aritméticas* y las *constantes de cadena (string)*. Veámoslas por separado.

2.3.1. Constantes aritméticas

Una constante aritmética en el SuperBASIC del QL representa un valor numérico que puede estar configurado de varias formas diferentes. Puede utilizarse cualquier número de dígitos en la construcción de constantes. Los números se representarán en forma *entera* o *real (de punto flotante).* Igualmente pueden representarse números utilizando la conocida notación científica o exponencial, que consiste en escribir un número entero o real (con o sin signo) seguido de la letra E y de una potencia entera de 10 (con o sin signo).

Seguidamente se muestran algunos ejemplos correctos de constantes aritméticas de punto flotante.

```
14E+2   (equivalente a 14E2 o a 1400)
4.75
36
−4.353535
+7.45
14E−2 (equivalente a 0.14)
10E6    (equivalente a 1000000)
```

El SuperBASIC no establece diferencia alguna entre las constantes enteras de punto fijo y las constantes reales de punto flotante, dado que todas ellas serán tratadas como de punto flotante prescindiendo de si poseen punto decimal o no. Por esta razón, las constantes 4 y 4.0 son equivalentes.

2.3.2. Las constantes de cadena (String)

Una *cadena (string)* en SuperBASIC es una secuencia de cero o más símbolos o caracteres. Dentro de un programa para el QL, una constante de string se representa por una cadena de caracteres encerrada entre comillas ("). Para la representación del carácter comilla dentro de una constante de string se utilizan las dobles comillas.

Así, para la representación de la cadena WX"YZ se escribirá:

```
"WX""YZ".
```

2.4. LOS IDENTIFICADORES

Un *identificador* en SuperBASIC es aquel nombre proporcionado por el usuario con el que se denotan o mencionan las variables del lenguaje.

Puede poseer una longitud máxima de 255 caracteres, siendo el primero de ellos obligatoriamente una letra y continuando cualquier número de dígitos o letras e incluso el signo de subrayado hasta una longitud máxima, como antes decíamos, de 255 símbolos.

Un posible diagrama sintáctico de los identificadores en SuperBASIC se muestra en la figura 2.2.

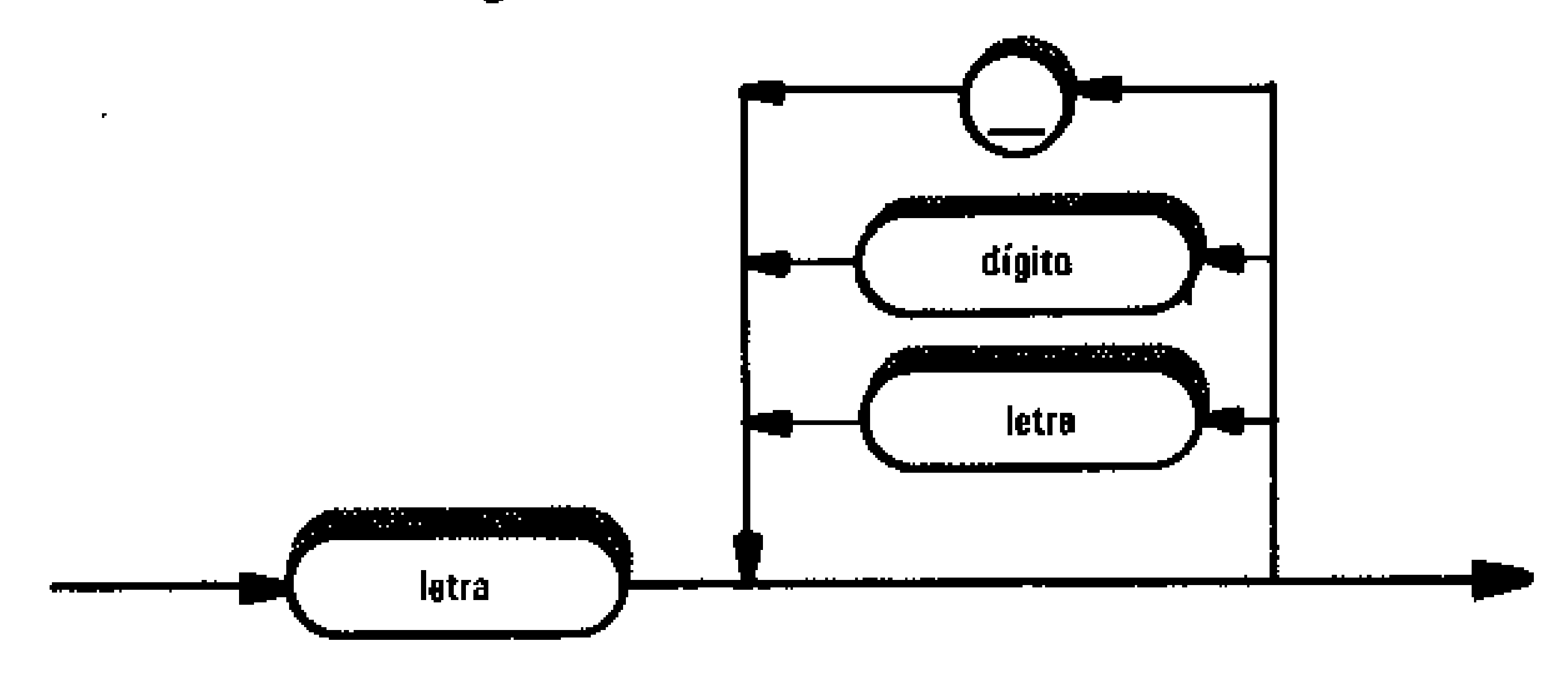


Fig. 2.2.— Diagrama sintáctico de un identificador.

Dependiendo del carácter con el que acabe el identificador, poseerá un sentido u otro para el SuperBASIC.

El cuadro de la figura 2-3 muestra las posibles alternativas lexicográficas de los identificadores.

Termina con	Significado
{ letra / dígito }	número real
%	número entero
$	cadena (string)

Fig. 2.3.— Terminaciones de los identificadores.

Hemos dicho que la barra de *subrayado (underscore)* puede ser utilizada como parte integrante de un identificador. Esto es particularmente útil para denotar con mayor precisión una variable.

Algunos ejemplos válidos de identificadores son:

```
calculo
pregunta
día_de_hoy
condición_final
días_365
```

No debe confundirse el símbolo de subrayado con el guión (hyphen), pues éste último carácter no está permitido en la composición de un identificador.

En el teclado del QL, el guión se encuentra en la zona baja de la tecla a la derecha del cero y el subrayado se encuentra en la zona alta. Así pues, para obtener el símbolo del subrayado bastará con apretar simultáneamente a éste la tecla SHIFT, o bien mantener activada la tecla CAPSLOCK que da acceso a los símbolos descritos en las zonas altas de cada tecla.

En SuperBASIC a semejanza con otros BASIC's (incluido el del Spectrum) no hay diferencia alguna en utilizar las letras mayúsculas o minúsculas para la escritura de los identificadores, pues se toman como idénticos.

Así, los siguientes identificadores SuperBASIC representan para el QL la misma variable:

```
vuelo_5a
VUELO_5a
Vuelo_5a
Vuelo_5A
vUElo_5a, etc.
```

El uso de los identificadores es una pieza fundamental para la comprensión y el desarrollo de los programas escritos en SuperBASIC pues su uso continuado proporcionará una mayor claridad a la hora de declarar procedimientos y/o funciones en vez de utilizar GOTOs o GOSUBs como ya veremos.

Por otro lado, una *variable real* en SuperBASIC puede tomar los valores comprendidos entre -10^{615} y $+10^{615}$ con solamente 8 dígitos significativos.

2.5. LOS OPERADORES

El SuperBASIC del QL puede utilizar dos tipos de operadores, ya sean relacionales o algebraicos. La tabla de la figura 2-4 muestra estos operadores y sus significados.

OPERADORES RELACIONALES

operador	denominación	aplicado a...
=	igual	números y cadenas
==	equivalencia	números y cadenas
<	menor que	números y cadenas
>	mayor que	números y cadenas
<=	menor o igual que	números y cadnnas
>=	mayor o igual que	números y cadenas
<>	no igual	números y cadenas

OPERADORES ALGEBRAICOS

operador	denominación	aplicado a...
+	suma	números
–	diferencia	números
/	división	números
*	multiplicación	números
&	concatenación	cadenas
&&	AND lógico	
||	OR lógico	
^^	OR exclusivo	
~~	NOT lógico	
OR	OR lógico	
AND	AND lógico	
XOR	OR exclusivo	
NOT	NOT lógico	
MOD	módulo	números
DIV	división	enteros
	elevación	reales
(^)	elevación	enteros
–	menos unario	
+	más unario	(NOP)

Fig. 2.4.– Relación de operadores de SuperBASIC.

Todos estos operadores disponen de una jerarquía de uso, esto es; de una cierta *prioridad* de ejecución de unos sobre otros.

La tabla de la figura 2-5 muestra el orden de jerarquía de dichos operadores de mayor a menor precedencia.

1	+, –	más o menos unario
2	&	concatenación de cadenas
3	^	exponenciación
4	*, /, MOD, DIV	multiplicación, división módulo y división entera.
5	+, –	suma y resta
6	Comparaciones lógicas	
7	NOT	
8	AND	
9	OR y XOR	

Fig. 2.5.– Tabla de procedencia de los operadores.

Utilizando la tabla anterior y sabiendo que a igualdad de jerarquía de operadores, estos se ejecutan de izquierda a derecha, intente el lector descubrir cuál será el resultado del segmento de programa:

```
40 a = 10 + 3 * 2
50 PRINT a                (1)
```

(1) Nota: Como ya se estudiará con detalle en el capítulo siguiente, adelantamos ahora que la sentencia PRINT lo único que hace es visualizar el contenido de la variable que se menciona.

Dado que el operador * de multiplicación posee mayor prioridad que el + de adición, el resultado de la ejecución del programa anterior dará 16.

No obstante, la prioridad de los operadores puede ser alterada con el uso de los paréntesis dentro de una expresión de cualquier tipo, de forma que se obligue a ejecutar lo encerrado entre paréntesis antes que cualquier otra operación de la expresión.

```
40 a = (10 + 3) * 2
50 PRINT a
```

En este caso, el resultado final será 26, dado que se efectúa primero 10 + 3 y después este resultado parcial se multiplica por 2.

Otra característica importante que nos brinda en SuperBASIC del QL es la posibilidad de manejar valores lógicos.

Así pues, si en una instrucción PRINT sobre expresiones de cualquier tipo que incluyan operadores relacionales del tipo >' = y < dicha relación resulta ser *verdadera (TRUE)*, entonces se imprimirá un *uno* (1). Si la expresión resulta ser *falsa (FALSE)*, entonces se imprimirá un *cero* (0).

Ejemplos:

```
10 a = 3
20 PRINT a = 2 + 1
dará como resultado: 1 (verdadero)

10 a = 3
20 PRINT 2 + 1 <> a
dará como resultado: 0 (falso)
```

La tabla de la figura 2-6 muestra algunos ejemplos del uso de los operadores aritméticos descritos con anterioridad.

Ejemplo:

Sea, por ejemplo, una baraja de cartas españolas donde cada carta se la hace corresponder un número de la forma:

1 al 10 as, dos, ..., rey de oros
11 al 20 as, dos, ..., rey de copas
21 al 30 as, dos, ..., rey de espadas
31 al 40 as, dos, ..., rey de bastos

Operación	Símbolo	Ejemplos	Resultado	Observaciones
Suma	+	5 + 50	55	
Diferencia	–	6 – 7.2	–1.2	
Multiplicación	*	4 * 3.1 2.4 * (–2)	12.4 –4.8	
División	/	9 / 2 –20 / 3 6 / 0	4.5 –6.666667 overflow	división por cero
Elevación a potencia	^	2^3	8	
Elevación a potencia (enteros)	(^)	3(^)2	9	sólo enteros
División entera	DIV	–8 DIV 2 9 DIV 2	–4 4	sólo enteros
Módulo (resto de la división)	MOD	13 MOD 5 –17 MOD 8	3 7	

Fig. 2.6.– Ejemplos de utilización de los operadores aritméticos.

El siguiente programa, dado un número comprendido entre 1 y 40, obtiene la denominación de la carta correspondiente. Aquel lector que en este momento encuentre alguna dificultad en comprender el programa, puede consultar algunos aspectos del mismo en el capítulo 5.

```
100 REMark identificacion de cartas
110 INPUT "teclee un numero ";carta
120 palo = (carta – 1) DIV 10
130 valor = carta MOD 10
140 IF valor = 0 THEN valor = 10
150 IF valor = 1 THEN PRINT "as de ";
160 IF valor >= 2 AND VALOR <8 THEN PRINT valor !"de"!
170 IF valor = 8 THEN PRINT "sota de ";
180 IF valor = 9 THEN PRINT "caballo de ";
190 IF valor = 10 THEN PRINT "rey de ";
200 IF palo = 0 THEN PRINT "oros"
210 IF palo = 1 THEN PRINT "copas"
220 IF palo = 2 THEN PRINT "espadas"
230 IF palo = 3 THEN PRINT "bastos"
```

Los signos de puntuación ! y ; en las instrucciones PRINT serán vistos con detalle en el capítulo 3.

2.6. EL CONCEPTO DE EXPRESION

Existen en SuperBASIC dos tipos de expresiones: *expresiones de cadenas* (string) y *expresiones algebraicas*. Veámoslas por separado.

2.6.1. Expresiones algebraicas

En SuperBASIC una expresión algebraica es cualquier combinación válida de constantes aritméticas, variables, funciones y operadores algebraicos.

Se muestran seguidamente algunos ejemplos de expresiones algebraicas correctas.

```
a
a + b
a1 + b * c / (H + i)
4 + b1 * INT (h + j –5.23)
f * (a – b ^ – c (p) + a) – 2.7181
etc.
```


Obsérvese que en el primer caso se ha escrito una expresión carente de operadores y que solamente posee una variable. Este es el caso más sencillo del concepto de expresión, pudiéndose utilizar una sola variable allí donde sintácticamente esté permitido el uso de expresiones numéricas.

2.6.2. Expresiones con cadenas

Una expresión con cadenas consta de cualquier variable de tipo string o de cualquier constante de tipo cadena.

Para la *concatenación* (1) de cadenas se utilizará el operador ampersand (&) y podrá ser utilizado en toda su extensión en la instrucción o sentencia de asignación (LET) como ya veremos más adelante.

Algunos ejemplos de expresiones con cadenas son:

```
"cadena"
a$
"a" & b$ (aplicada a la sentencia de asignación)
```

2.7. NOTACION EMPLEADA

A lo largo de este texto utilizaremos una notación ya muy extendida para la definición sintáctica de lenguajes de programación y que emplearemos en los formatos de las instrucciones y comandos SuperBASIC que estudiemos.

Resumamos estas reglas ahora:

1) Las palabras que aparezcan en **negrita** son palabras-reservadas del SuperBASIC y no pueden emplearse fuera de su contexto de definición. En el Apéndice-C se encuentra una lista completa de tales palabras y un estracto de su función.

2) Estas palabras-reservadas pueden estar escritas íntegramente con letras mayúsculas o parte con mayúsculas y parte con minúsculas. Esto significa que solamente aquella porción escrita con mayús-

---

(1) Nota: La concatenación de cadenas, como veremos más adelante, es una operación que "une" los símbolos o caracteres de cada cadena mencionada.
Así:

"abc" & "de" dará como resultado la cadena "abcde"

culas es *necesaria* para escribir la palabra, pudiendo omitirse el resto.

3) Todas las palabras restantes de un formato son las suministradas por el usuario y estarán escritas habitualmente en letra *cursiva.*

4) Cuando ciertas palabras se encuentren encerradas entre paréntesis cuadrados [ ] significa que lo de dentro es opcional, esto es; que puede escribirse o no.

5) Los caracteres de puntuación (puntos, comas, etc) representan la aparición real de tales caracteres.

6) Cuando ciertas palabras se encuentran encerradas entre *llaves* { } significa que existen posibilidades diferentes y representan la aparición obligatoria de una y sólo una de esas posibilidades.

7) Cuando estas llaves se encuentren *elevadas* a una potencia *n*, $\{\ \}^n$ se quiere representar la posible aparición de hasta *n-veces consecutivas* de uno de los datos interiores.

2.8. LOS COMENTARIOS (REMARK)

Los comentarios en SuperBASIC se utilizarán dentro de un programa cuando se desee explicar algo acerca del mismo. Los comentarios no tendrán efecto alguno sobre el intérprete de SuperBASIC y la línea, a la hora de la ejecución, que contiene el comentario será ignorada.

El formato de una instrucción REMark es:

REMark [*texto*]

donde *texto*: puede ser cualquier secuencia de caracteres y puede escribirse o no.

Ejemplos

```
10   REMark Aqui comienza el programa
20   FOR i = 1 TO 10
30      PRINT i: REMark  Se imprime i
40   END FOR i
50   STOP: REMark final del programa
```


Cuando la palabra REM es la primera de una línea, el resto de la línea será omitida, en tiempo de ejecución, por el QL, como en el caso de:

```
100  REMark final del programa: PRINT "final"
```

La instrucción PRINT anterior jamás podrá ejecutarse.

Sea ahora el momento de recordar que el SuperBASIC del QL reconocerá cualquier comando o sentencia escrita con letras mayúsculas o minúsculas indistintamente. No obstante, como ya mencionábamos antes, algunas de las palabras reservadas que utilizaremos a lo largo del texto estarán escritas con mayúsculas solamente en alguna porción de las mismas. Con esto queremos significar que dichas letras son las exclusivamente necesarias y suficientes para escribir la mencionada palabra-clave.

Ejemplos:

REMark
REPeat
SELect
DEFine PROCedure
etc.

2.9. LA SENTENCIA DE ASIGNACION (LET)

Cuando se desea que una variable tome el resultado de una expresión, ya sea algebraica o de string, se escribe en SuperBASIC una sentencia de asignación.

El valor de la expresión se calcula y se deposita en la variable especificada.

El formato general de una sentencia de este tipo es:

[LET] variable = expresión

Sean algunos ejemplos de sentencias de asignación:

```
10   LET a (1) = b * (K + c1)
25   LET nom$ = "galan"
40   LET ql$ = "super" & "basic"
```

El modo de la expresión debe ser compatible con el modo que posea la variable receptora de la sentencia de asignación. No obstante el SuperBASIC dispone de una característica muy importante y que hemos llamado *compatibilidad* o *conversión (coerción)* que hace que se puedan mezclar operandos y operadores no-homogéneos dentro de la misma sentencia de asignación. Esta ventajosa peculiaridad del QL será estudiada con detalle en el apartado siguiente.

En SuperBASIC del QL puede omitirse en la sentencia de asignación la palabra reservada LET, tal y como se desprende de la observación del formato anterior.

La sentencia de asignación se utilizará también cuando se desee llevar el contenido de una variable a otra.

Ejemplos:

```
10   a$ = b$
30   suma = total
etc.
```

Obsérvese que en las instrucciones anteriores se ha omitido el uso de la palabra LET.

Es muy importante hacer notar al lector que el uso del símbolo = dentro de una sentencia de asignación *no* posee el mismo significado que en matemáticas. Por eso, en SuperBASIC podemos escribir.

```
10 LET x = x + 1
```

sin ningún inconveniente, pues lo único que se hace es llevar el resultado de sumar x y 1 de nuevo a x.

Respecto de los identificadores enteros (los que terminan con %) y de la sentencia de asignación puede decirse que:

```
100  LET entero% = 7.45
```
almacenará en la variable entero % el valor 7, pero

```
150  entero% = 7.64
```
almacenará en dicha variable el valor 8.

Es decir, la asignación de una expresión real a una variable entera hace que se *redondee* dicho valor a su entero más próximo.

2.10. LA CONVERSION O COMPATIBILIDAD (COERCION)

Una característica muy importante del SuperBASIC que no poseen otros BASIC's es la compatibilidad entre los operandos de una expresión, que se obligan a ser tratados de la misma forma aún cuando sean objetos de tipo heterogéneo.

Observemos el segmento de programa siguiente:

```
10   PRINT "teclee su edad"
20   INPUT edad
30   PRINT "teclee el día del mes"
40   INPUT mes$
50   PRINT "la suma es ="; edad + mes$          (1)
```

En la línea número 10 se está pidiendo al usuario que introduzca un número que se depositará en la variable numérica "edad". Igualmente en la línea número 30 se le vuelve a indicar que introduzca otro número que se depositará en la variable de cadena mes$. En la línea 50 se suman aritméticamente los contenidos de ambas variables.

Esta circunstancia daría seguramente un error y provocaría que el programa fallase al intentar la ejecución de la línea 50 en muchos de los lenguajes BASIC conocidos, dado que se está intentando sumar cantidades que no son compatibles como son las variables reales y los strings.

Sin embargo en el QL la característica de *conversión (coerción)* hace que este segmento de programa no plantee ninguna dificultad de ejecución de forma que ambos números serán sumados como si de cantidades extrictamente aritméticas se tratara.

Así, cuando, por ejemplo, se teclea 24 en la línea de programa 40, el SuperBASIC la almacenará en la variable mes$ como "24", es decir, como string y no como número. Será en el momento de la suma edad *X* mes$ cuando *convertirá* éste "24" a simplemente 24 con el objeto de que sea posible su suma con el valor que pudiera contener la variable edad.

(1) Nota: Ya será estudiada con detalle en el capítulo siguiente la instrucción INPUT. Baste ahora con decir que tal sentencia toma o recoge valores del exterior (por ejemplo, desde el teclado) y los introduce en las variables mencionadas.

Así pues, y con el fin de evitar fallos en tiempo de ejecución, el SuperBASIC del QL intentará la compatibilización de los operadores de una expresión numérica o de string siempre que esto sea posible.

Como norma general podemos afirmar que serán transformados todos aquellos valores de una expresión de acuerdo con el *tipo de operador* que se escriba en la misma.

Veamos otro ejemplo en el siguiente segmento del programa:

```
10   PRINT "en que calle vive? "
20   INPUT calle$
30   PRINT "en que numero? "
40   INPUT numero
50   LET direccion$ = calle$ & numero
60   PRINT direccion$
```

Este sería otro programa que nos daría problemas de ejecución en la mayoría de los BASIC's conocidos.

Obsérvese como en la línea 50 se está intentando concatenar una cadena (calle$) con un valor numérico (número).

En este caso el SuperBASIC compatibilizará el valor número como si de un string se tratara, puesto que es el símbolo & de concatenación de cadenas el que está presente.

Una posible ejecución del programa anterior podría ser:

```
en que calle vive?
CASTELLANA
en que numero?
106
CASTELLANA106
```

En este programa, 106 se almacena como un número en la variable número pero al llegar a la línea 50 del programa se transforma en "106" de forma que se convierte en una cadena y puede ser concatenada con el contenido de calle$ sin más que observar que el tipo de operador que los une es para cadenas (&).

Diremos, por tanto, que la conversión de los elementos de una expresión, ya sea aritmética o de string, vendrá dada y será posible siempre que sean del mismo tipo el operador inmerso en la expresión (en este último caso el &) y la variable receptora del resultado (en este caso la dirección$).

Los siguientes ejemplos, son algunos de la característica de conversión (coerción) entre operadores.

Ejemplos:

```
LET a = "365" + 34
que dará como resultado: 399

LET a = "1" + "2" + "3"
que dará como resultado: 6

LET a$ = 123 & 456
que dará como resultado: "123456"

LET a$ = "123" & 45
que dará como resultado: "12345"
```

La conversión o coerción entre operadores no es necesaria cuando se pretenden sumar dos variables reales de punto flotante.

Recordemos que una variable numérica de punto flotante se designa por cualquier combinación de letras y de números comenzando siempre por una letra (p. ej. a257b, h342, z123, etc.).

No obstante, si lo que se pretende es sumar dos variables con contenido real de punto flotante y depositar dicha suma en una variable entera, tampoco es necesaria la conversión de ambas cantidades flotantes aunque el resultado de la suma se almacenará como un número entero.

Recordemos que una variable entera se denota por cualquier combinación de letras y de dígitos, comenzando por una letra y terminando con el símbolo

Así pues, del uso de los operadores + y & y del tipo de la variable donde alojar el resultado, se tendrá la posible transformación o compatibilización de los operadores de una expresión. Es por esta razón por la que se recomienda un uso cuidadoso de ambos operadores con el fin de evitar resultados no deseados.

Trate el lector ahora de decidir cuáles serían los resultados de los dos siguientes programas SuperBASIC.

Ejercicios:

```
(1)     10  a$ = "1234.56"
        20  b = 78.9
```

```
        30  c$ = a$ + b
        40  PRINT c$

(2)     10  a$ = "1234.56"
        20  b = 78.9
        30  c$ = a$ & b
        40  PRINT c$
```

En el programa (1) en la línea 30 se está indicando con el signo + que deben sumarse aritméticamente dos operandos que son "1234.56" (de tipo string) y 78.9 (de punto flotante). Luego la cadena "1234.56" será convertida o transformada a numérica para que dicha adición aritmética pueda tener lugar, de forma que el resultado final será "1313.45" (en forma de string) dado que la variable receptora es c$ que es de tipo cadena.

En el programa (2) en la línea 30 se está indicando con el signo & que deben concatenarse dos operandos que son "1234.56" (de tipo string) y 78.9 (de punto flotante). Luego el número 78.9 será convertido o transformado a string para que dicha concatenación pueda tener lugar, de forma que el resultado final será "1234.5678.9" (en forma de string) dado que la variable receptora es c$ que es de tipo cadena.

Resumiendo, podemos afirmar que el signo + se utilizará por el QL cuando se pretendan tratar a las variables como números y el signo & se utilizará cuando se pretenda tratar a las variables como cadenas de caracteres (string).

Existe un orden natural sobre los diferentes tipos de datos dentro del SuperBASIC del QL. La figura 2.7 representa los tipos más generales de datos y sus posibles conversiones.

Las flechas de la figura anterior muestra la factibilidad de la conversión (coerción). Así, se observa como siempre es posible establecer una compatibilidad de los datos si se va ascendiendo de los niveles más particulares a los niveles más generales, pero no siempre es posible lo contrario.

Ejemplo:

– campo1 = campo2 + campo3
  No es necesaria la conversión ni antes ni después de ejecutar la suma.

– a = campo2 + campo3
  Antes de ejecutar la suma no es necesaria la conversión, pero sí después, ya que el resultado se convertirá a entero antes de la asignación.

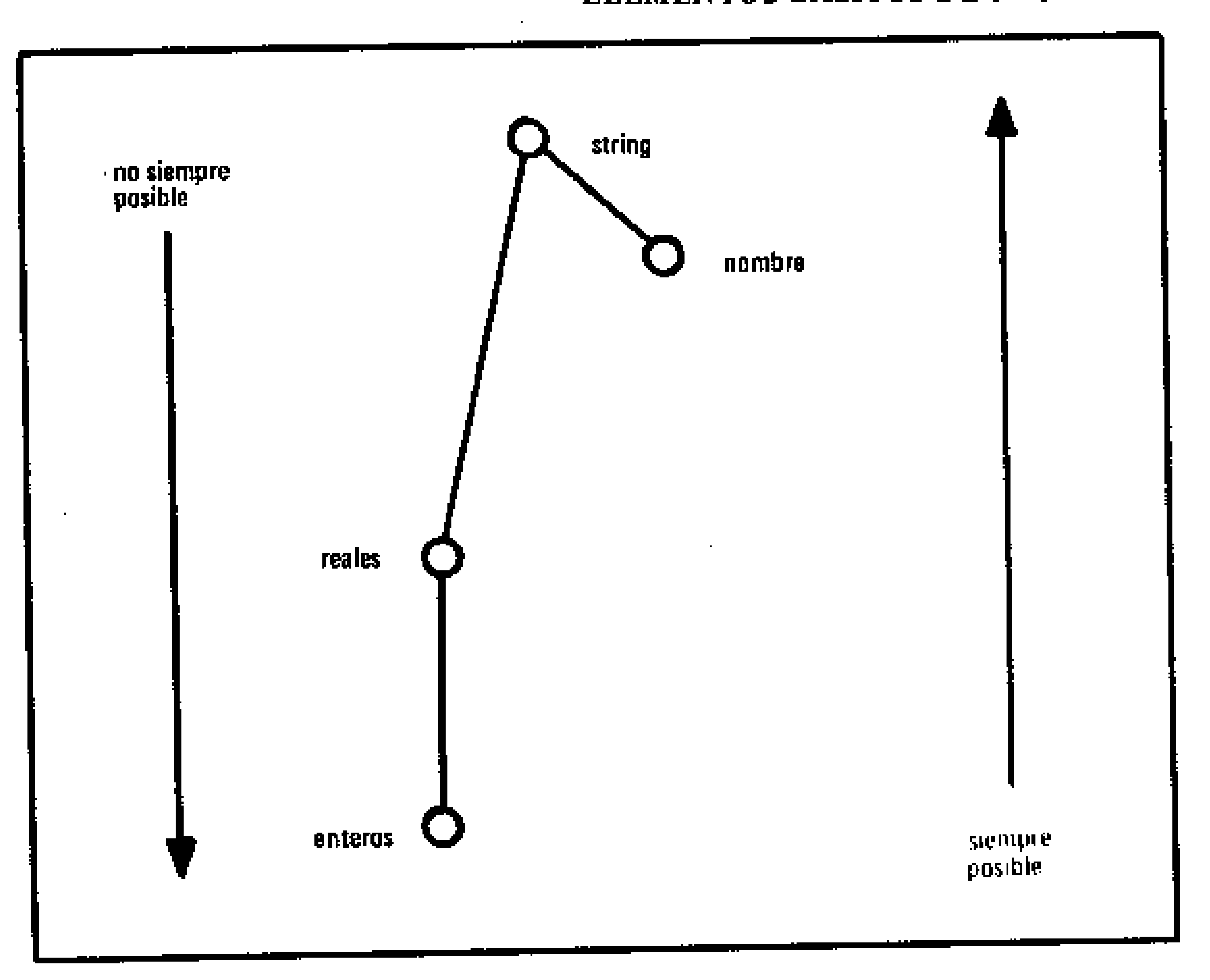


Fig. 2.7.– Tipos de datos y sus posibles secuencias de conversión.

– a$ = b$ + c$
  Las variables b$ y c$ serán convertidas a cantidades reales, si esto es posible, antes de realizar la suma. El resultado se convertirá a tipo string antes de la asignación.

– **SAVE** "mdv1–fichero"
  La cadena anterior será convertida a tipo *nombre* para que actúe con el procedimiento SAVE antes de que sea usada.

2.11. LAS INSTRUCCIONES READ Y DATA

La instrucción READ se utiliza para asignar valores a determinadas variables.

El formato general de una instrucción READ es:

READ $v_1, v_2, \ldots, vn$

La sentencia READ actúa emparejada con la instrucción DATA que es la que se encarga de señalar *con qué* valores han de cargarse las variables de la instrucción READ.

El formato general de una sentencia DATA es:

DATA $c_1, c_2, ..., c_n$

donde c_1, c_2, etc., representan constantes de tipo numérico o de string e incluso expresiones.

El funcionamiento conjunto de ambas instrucciones es el siguiente: cada vez que el programa encuentra una instrucción READ, las variables aritméticas y las variables de string especificadas son asignadas con los datos que aún quedan sin asociar y que estén escritos en una instrucción DATA.

Esto se hará de tal forma que siempre que se ejecute una instrucción READ deberá verificarse que aún quedan datos por leer de su anexa instrucción DATA. En caso contrario, el programa terminará su ejecución.

Sea, por ejemplo, el siguiente programa:

```
10   DATA 17, 45, 61, 85, 92
20   READ a
30   PRINT a, 2 * a
40   GOTO 20
```

Como hemos mencionado antes, la sentencia DATA proporciona una lista de valores que se usarán en el programa de forma que estos valores serán accesibles por una o más sentencias READ.

En el programa anterior, la primera sentencia que se ejecuta es la correspondiente a la línea 20, esto es; la READ, por tanto, el primer valor que se le asigna a A es 17 imprimiendo después dicho valor y su doble. Después el control vuelve a la sentencia READ que toma la primera de las constantes de la lista DATA que aún queda por leer, siendo el valor 45 que se vuelve a asignar a A y se imprime, etc. Cuando se lean todos los valores de la sentencia DATA, el programa parará su ejecución.

Las instrucciones DATA pueden aparecer en cualquier lugar del programa y puede haber tantas sentencias DATA como sean necesarias.

Así pues, el programa:

```
10   DATA 17,45
20   DATA 61
30   READ a
40   PRINT a, 2 * a
50   DATA 85,92
60   GOTO 30
```

es completamente equivalente al anterior.

Lo más común, no obstante, es que todas las instrucciones DATA de un programa se encuentren agrupadas todas juntas, o bien al comienzo del programa o bien inmediatamente antes del final del mismo.

2.12. LA INSTRUCCION RESTORE

La sentencia RESTORE se utiliza conjuntamente con las instrucciones READ y DATA en un programa y sirve para poder *reutilizar* una lista de datos de una sentencia DATA.

Cuando el programa se encuentra con una instrucción RESTORE todas las constantes escritas en cualquier sentencia DATA pueden volver a ser utilizadas de forma que la siguiente instrucción READ que sea ejecutada leerá el primer componente de la primera sentencia DATA del programa.

El formato general de una sentencia de este tipo es:

RESTORE [*línea*]

Cuando una instrucción RESTORE va acompañada de un número de *línea*, entonces esta instrucción solamente se referirá a los datos contenidos en la instrucción de la línea mencionada.

Es importante hacer notar al lector que en el momento de la ejecución de un programa no se presupone a priori la ejecución implícita de ninguna instrucción RESTORE. Por este motivo, si quiere ejecutarse el programa repetidas veces es necesario escribir explícitamente una sentencia RESTORE o CLEAR antes de proceder a ejecutar el programa.

3

Instrucciones de entrada/salida

3.1. INTRODUCCION

En este capítulo se estudiarán con detalle las instrucciones de entrada/salida habituales del SuperBASIC como son:

la sentencia INPUT
la sentencia PRINT

y otras instrucciones para su manejo conjunto con ambas.

Nos centraremos fundamentalmente en la entrada y salida de datos a través del teclado y de pantalla, respectivamente, dejando para el capítulo 10 las entradas y salidas correspondientes al manejo y utilización de ficheros.

3.2. LA INSTRUCCION INPUT

La sentencia INPUT se utilizará para llenar de contenido una o varias variables, del tipo que sean, desde el exterior.

El formato más habitual de una instrucción INPUT es:

INPUT	[texto,]	$v_1, v_2, ..., v_n$

La sentencia INPUT es similar en cuanto a la forma y a la función que la sentencia READ; sin embargo, en vez de utilizar la instrucción DATA como fuente de la lista de datos que hay que leer, la instrucción INPUT requiere los datos en tiempo de ejecución del programa, esperando que el usuario le suministre la información necesaria para llenar las variables escritas en la propia instrucción INPUT.

Sea, por ejemplo, la instrucción siguiente:

```
10   INPUT  a, b (1), a$, s$ (n)
```

Cuando se llegue a la ejecución de esta sentencia, el usuario deberá introducir por el teclado del QL las informaciones con que desea que se inicialicen las variables mencionadas; esto es, dos de tipo numérico y dos de tipo string.

El usuario deberá escribir sus datos de forma que se correspondan uno a uno y posicionalmente con las variables mencionadas en la instrucción INPUT.

La primera constante que escriba será, por tanto, el valor que tomará "a", el segundo será el valor que tomará "b(1)" etc.

Algunos otros ejemplos de la utilización de la sentencia INPUT con un *texto* asociado que naturalmente irá entre comillas, son:

Ejemplos:

```
10   INPUT  "Escriba su nombre"!nombre$
30   INPUT  "numero?";numero
60   INPUT  ("elemen" & elemen)!tabla (e)
```

Los separadores , ; y ! serán estudiados con detalle en el siguiente apartado.

3.3. LA INSTRUCCION PRINT

Ya hemos utilizado en alguno de los ejercicios anteriores la sentencia PRINT. Veámosla ahora con detalle.

El formato más común de la instrucción es:

PRINT	[*canal,*]	$p_1, p_2, ..., p_n$	(1)

(1) Nota: El término *canal* utilizado en este formato y en varios más adelante será visto con posterioridad en los capítulos 9 y 10 y por esta razón no se explica ahora.

donde los p_i pueden ser constantes, variables de cualquier tipo o cadenas de caracteres encerrados entre comillas o incluso expresiones.

Los separadores (que en el formato son comas) pueden ser cualquiera de los siguientes símbolos con el significado especial:

- , salida tabulada (en forma de columnas de impresión)
- ; sin separación entre campos.
- \ obliga a cambiar a una nueva línea.
- ! proporciona un espacio en blanco entre cada campo visualizado.

Ejemplos:

```
10   PRINT 1,2,3
visualizará:        1     2     3

10   PRINT 1!2!3
visualizará:        1 2 3

10   PRINT 1\2\3
visualizará:        1
                    2
                    3

10   PRINT 1;2;3
visualizará         123

10   PRINT  "Esto es un TEXTO"
visualizará:        Esto es un TEXTO

10   nombre$="carlos"
20   PRINT nombre$!"galan"
visualizará:        carlos galan
```

La sentencia PRINT posee además una variante que permite visualizar un texto o valor en una determinada columna del canal seleccionado (pantalla, impresora, etc.).

El formato de esta variante es:

PRINT [*canal*] **TO** *expresión-numérica*

donde *expresión-numérica:* debe representar un valor de columna razonable para visualizar el dato. Si esto no fuera así, esta instrucción no tendrá efecto.

Ejemplos:

```
20   PRINT TO 15;"Esto esta en la columna 15"
40   PRINT TO i;"sera en la columna i-esima"
```

El SuperBASIC del QL posee varias instrucciones que permiten visualizar informaciones sujetas a una instrucción PRINT en cualquier lugar de la pantalla. Veámoslas.

3.3.1. La instrucción AT

El formato de la instrucción es:

AT *columna,línea*

Una instrucción AT puede ir seguida perfectamente por una sentencia PRINT.

Ejemplo:

```
10   AT 12,14: PRINT "columna 12, fila 14"
```

3.3.2. La instrucción CURSOR

El formato general de la instrucción es:

CURSOR [*canal,*] *x, y* [*,x,y*]

Esta instrucción se utilizará para posicionar el cursor dentro de la pantalla, ya sea en modo 256 ó 512 pixels.

Ejemplo:

```
30   CURSOR 100,150: PRINT "abcisa 100, ordenada 150"
```

Cuando se utiliza esta instrucción con cuatro parámetros, entonces las primeras x, y representan el origen de coordenadas (usando el sistema gráfico de coordenadas) y el segundo par x, y representa la posición del cursor (en el sistema de pixels coordenados) relativo al punto anterior.

Ejemplo:

```
30  CURSOR 40,40,15,15
```

tomará como origen de coordenadas el punto (40,40) y situará el cursor en el punto (15,15) relativo al punto anterior.

4
Matrices y cadenas

4.1 INTRODUCCION

En este capítulo se estudiarán con detalle dos conceptos fundamentales para la programación en SuperBASIC: *las matrices* y *las cadenas* o *strings*. Ambos conceptos son de importancia decisiva en la realización de programas y serán estudiadas todas aquéllas instrucciones del lenguaje de programación como la DIMensión, la DIMN, las funciones de codificación de los caracteres manejados, la función CODE, el manejo y tratamiento de cadenas de datos, la característica de la fragmentación (slicing) y en fin todas aquellas sentencias para la programación de estos dos tipos de estructuras de datos tan importantes en cualquier lógica de programación.

4.2. EL CONCEPTO DE MATRIZ O TABLA

Hasta ahora, en los capítulos anteriores, hemos visto como se utilizaban en SuperBASIC datos numéricos ya fueran enteros o reales (de punto flotante). En el presente capítulo vermos cómo declarar y utilizar una nueva estructura de datos llamada genéricamente *matriz* y cómo utilizar las cadenas de caracteres o strings como si de matrices se tratara.

Para el lector no introducido en el terreno matemático o de programación, diremos que una *matriz* es un conjunto de *elementos* o *componentes* todos del mismo tipo y que, agrupados, dan lugar a un área de memoria que puede ser tratada con un carácter independiente y de una forma especial.

Debemos mencionar que los elementos de los que está compuesta una matriz se encuentran ordenados, no por sus contenidos, pero si por su posición relativa dentro de la propia matriz.

Las matrices pueden ser de *una dimensión* (en cuyo caso también se las denomina *vectores*) o de *varias dimensiones.*

La figura 4.1 muestra un esquema de cómo puede representarse una matriz de una dimensión.

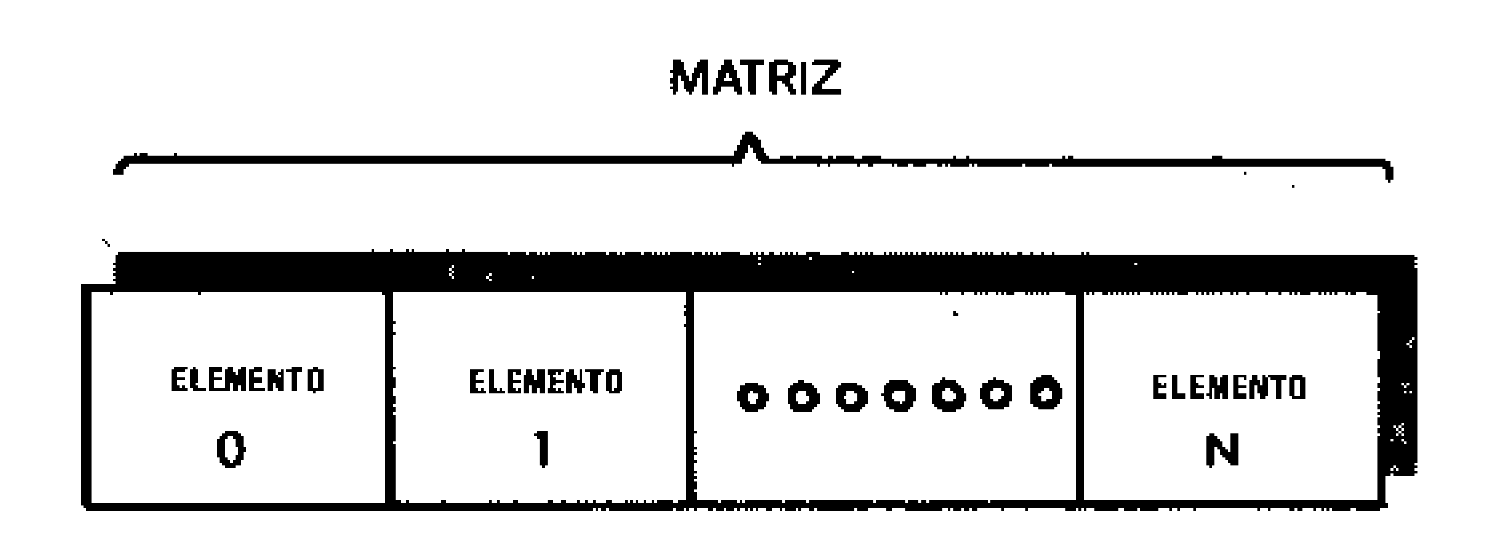


Fig. 4.1.— Representación de una matriz de una dimensión.

En la figura 4.1 puede observarse como el identificador MATRIZ da nombre a todo un área de memoria que se encuentra dividida en *n elementos o componentes* cada uno de ellos posee un número que lo identifica con claridad de los restantes elementos.

La utilización de matrices para almacenamiento de datos es muy habitual en la programación práctica con cualquier lenguaje.

Así, por ejemplo, si tuviéramos la necesidad de guardar en nuestro programa las siglas de las matrículas provinciales de los automóviles de España podríamos muy bien utilizar tantas variantes como necesitáramos. Esto es:

```
10   a0$ = "A  ": REMark Alicante
20   a1$ = "AB": REMark Albacete
30   a2$ = "AL": REMark Almería
.............................
500  a49$ = "ZA": REMark Zamora
```

Pero ésta forma de declaración, como el lector puede observar, es muy tediosa, cuanto más si necesitáramos manejar mayor número de elementos.

Así pues, si dispusiéramos de una estructura de datos como la mostrada en la figura 4.2 la utilización sería mucho más sencilla y sobre todo, mucho más lógica.

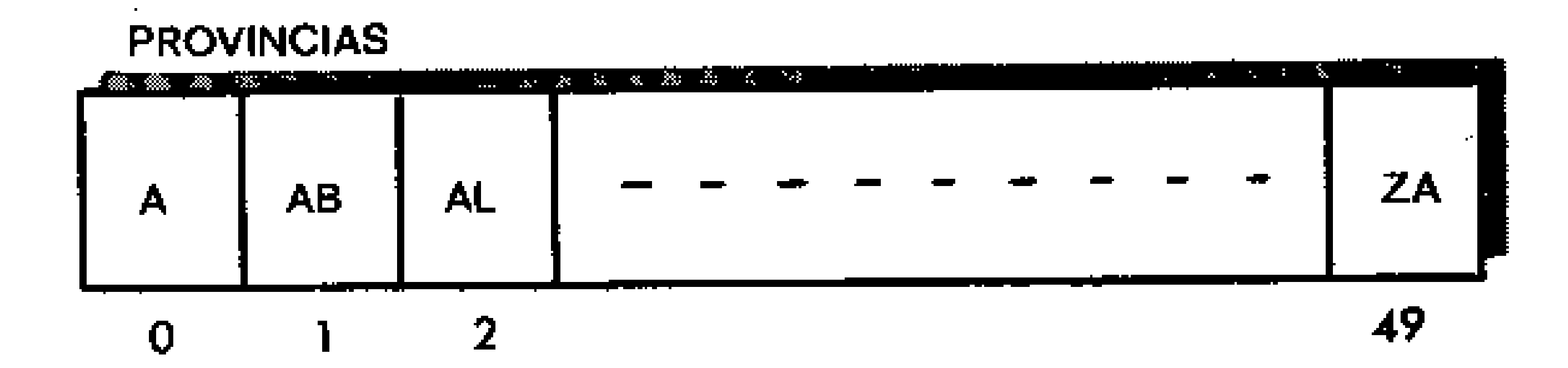


Fig. 4.2.— Ejemplo de almacenamiento de una matriz de una dimensión.

En el ejemplo anterior dispondríamos de un área de memoria llamada "PROVINCIAS" que constaría de 50 elementos numerados del 0 al 49, cada uno de ellos referenciable por su número y que contuvieran las siglas de las matrículas.

4.3 LA SENTENCIA DIMENSION

Para la declaración en SuperBASIC de las matrices se utiliza la sentencia DIMensión.

El formato de una sentencia de este tipo es:



A cualquier elemento de una matriz en el QL se le puede hacer referencia proporcionando el *nombre* de la matriz y un *subíndice*, que indica, como hemos visto, la posición del elemento dentro de la matriz.

Sea la siguiente declaración SuperBASIC de una matriz

```
10   DIMension matriz(5)
```

Con esta instrucción se indica que se va a utilizar una matriz de 6 elementos (numerados del 0 al 5).

Cuando se declara cualquier matriz, los subíndices pueden tomar los valores desde 0 hasta el máximo especificado en la instrucción DIMensión de forma que se genera siempre un elemento más (el cero) en cada dimensión que el propio número escrito en la declaración.

Las reglas de construcción de los nombres de las matrices siguen las mismas observaciones que las hechas para los identificadores. En la de-

claración anterior MATRIZ es un identificador numérico de punto flotante; por consiguiente cada uno de los 6 elementos de esta matriz serán números de este tipo.

Cuando se declara una matriz de tipo numérico, cada uno de sus elementos se inicializa automáticamente con ceros. De análoga manera cuando se declara una matriz de tipo string cada uno de sus elementos se inicializa automáticamente con cadenas de longitud cero.

La referenciación a uno cualquiera de los elementos de la MATRIZ anterior se realiza indicando entre paréntesis el subíndice del elemento referenciado.

Como hemos dicho, en el momento de la declaración de una matriz con la sentencia DIMensión se rellenan automáticamente cada uno de sus elementos con ceros (en el caso de que la matriz sea numérica).

Así pues, la ejecución del programa:

```
10   DIMension matriz(2)
20   PRINT matriz(0)
30   PRINT matriz(1)
40   PRINT matriz(2)
```

daría como resultado:

```
0
0
0
```

Si quisiéramos llenar esta matriz con los valores de tres números cualesquiera podríamos ejecutar el siguiente programa:

```
10   DIMension matriz (2)
20   matriz(0) = 10
30   matriz(1) = 283.32
40   matriz(2) = 45.7
```

Para referirnos al elemento que contiene 283.32 bastaría que mencionáramos MATRIZ(1). Nuestro programa podría manipular posteriormente este elemento MATRIZ(1) como si de una variable numérica cualquiera se tratase.

Este tipo de matriz como el que acabamos de describir corresponde a lo que se entiende como matriz de *una dimensión* y sirve, en el caso anterior, para contener una lista de números.

Sucede, no obstante, con gran frecuencia, que se necesita disponer de matrices de dos o más dimensiones para el almacenamiento de datos.

Sea, por ejemplo, el cuadro siguiente:

PRODUCCION DE VEHICULOS (en miles de unidades)				
clase \ año	1985	1986	1987	1988
Turismos	154	249	275	303
Motocicletas	99	100	60	40
Industriales	67	81	88	79

En el cuadro anterior puede observarse que para cada uno de los años mencionados (1985, 1986, 1987, 1988) existen tres tipos de vehículos (turismos, motocicletas e industriales) que arrojan un total de 12 cantidades que es necesario almacenar.

Pues bien, la forma más habitual de representación de matrices de *dos dimensiones* es la *construcción rectangular* como se muestra en la Figura 4.3.

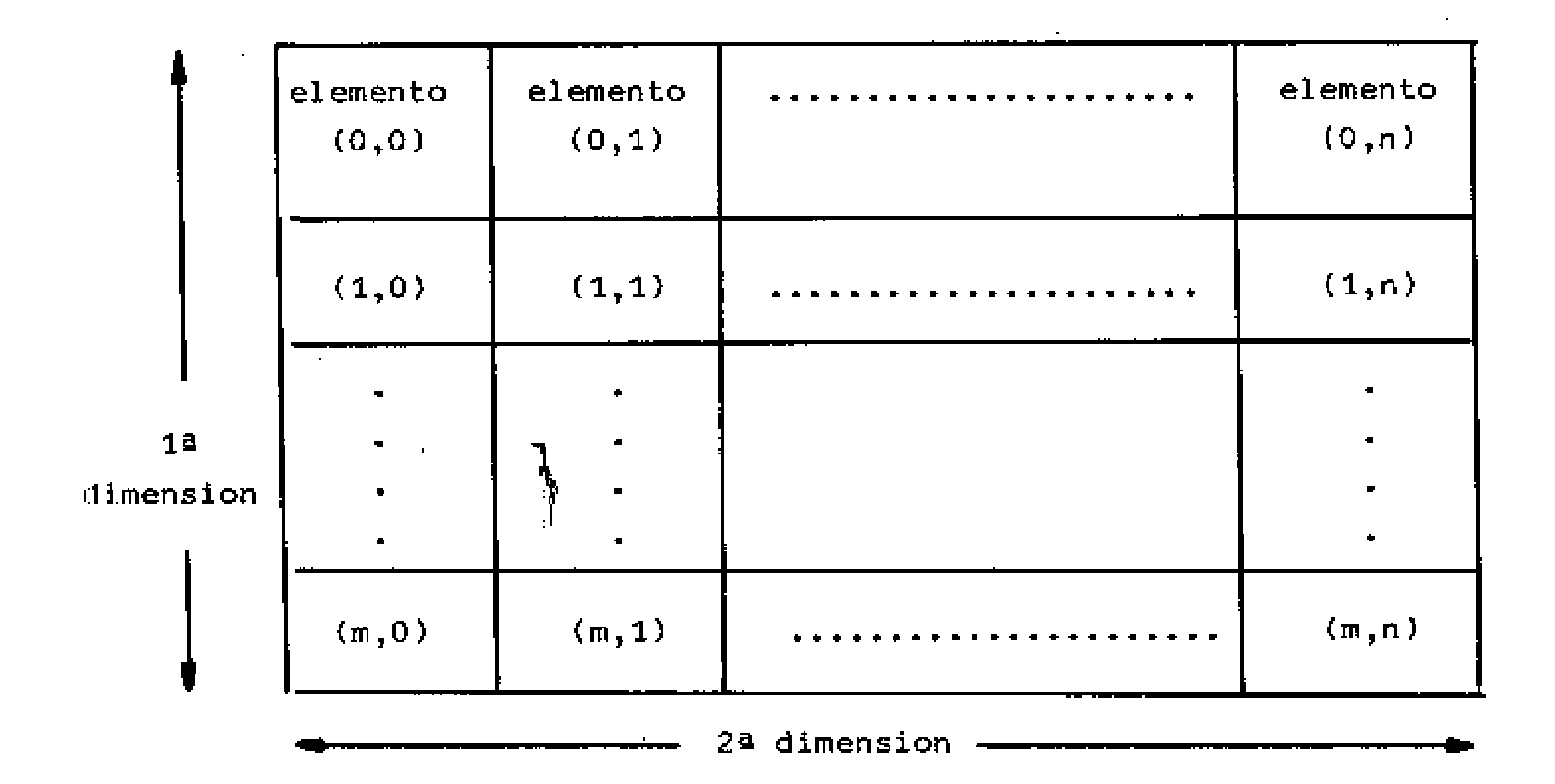


Fig. 4.3.— Representación de una matriz de dos dimensiones.

En la figura 4.3 puede observarse que, a diferencia de las matrices de una sola dimensión que poseían un sólo subíndice; en las matrices de dos dimensiones ya son necesarios *dos subíndices* para acceder a cada elemento.

Como norma general diremos que para referenciar a un elemento de una matriz de *n-dimensiones* son necesarios *n-subíndices.*

La estructura en forma de matriz de dos dimensiones del cuadro de VEHICULOS anterior podría ser la mostrada en la figura 4.4.

VEHICULOS

	0	1	2	3
0	154	249	275	303
1	99	100	60	40
2	67	81	88	79

Fig. 4.4.– Ejemplo de almacenamiento de una matriz de dos dimensiones.

Así pues, con la utilización de las parejas de subíndices para la primera dimensión *(las filas)* y para la segunda dimensión *(las columnas)* podríamos referenciar cada una de las cantidades del cuadro anterior.

Dado que las matrices de una o varias dimensiones ocupan mucho espacio de memoria en el QL se recomienda al lector un uso cuidadoso de tales características respecto del número y tamaño de las dimensiones.

Como norma general una matriz multi-dimensional se declara con la adición de un segundo (o tercero, etc) número después del primero y separado por comas.

Así, la declaración de la matriz siguiente:

```
10    DIMension doble(2, 3)
```

dimensiona una matriz de 3 por 4 elementos (no nos olvidemos que se empieza a contar desde cero).

La matriz VEHICULOS de la figura 4.4 podría ser declarada y llenada con el siguiente segmento de programa.

```
10    DIMension vehi(2, 3)
20    vehi(0, 0) = 154
```



```
30    vehi(0, 1) = 249
40    vehi(0, 2) = 275
50    vehi(0, 3) = 303
60    vehi(1, 0) =  99
70    vehi(1, 1) = 100
etc.
```

4.4 MATRICES DE LITERALES

Imaginemos la declaración de la matriz:

```
10    DIMension mat(3)
```

Una forma ya vista de llenar esta matriz con valores sería:

```
20    mat(0) = 10
30    mat(1) = 88.3
40    mat(2) = 983.4
50    mat(3) = 7
```

Existiría, como es sabido, otra forma tradicional de llenado de la matriz utilizando las instrucciones READ y DATA de la forma:

```
10    DIMension mat(3)
20    DATA 10, 88.3, 983.4 , 7
30    READ mat(0), mat(1), mat(2), mat(3)
```

No obstante, el SuperBASIC del QL nos permite el llenado de una matriz de una forma más adecuada e incluso más clara, utilizando *conjuntos de literales* encerrados entre llaves ({ }).

El siguiente programa muestra como podría realizarse tal llenado de la matriz anterior.

```
10    DIMension mat(3)
20    mat = {10, 88.3, 983.4, 7}
30    PRINT mat(0)
40    PRINT mat(1)
50    PRINT mat(2)
60    PRINT mat(3)
```

El resultado de la ejecución del programa anterior daría como resultado:

```
     10
   88.3
  983.4
      7
```

Como ha podido observarse en la línea 20 del programa anterior se ha procedido al llenado de la matriz MAT utilizando las facilidades que nos brinda el SuperBASIC para estos cometidos.

De análoga manera podremos llenar matrices de varias dimensiones.

```
10   DIMension doble(1, 1)
20   doble = {{1,2}{3, 4}}
30   PRINT doble(0, 0)
40   PRINT doble(0, 1)
50   PRINT doble(1, 0)
60   PRINT doble(1, 1)
```

El resultado de la ejecución del programa anterior dará como resultado:

```
1
2
3
4
```

4.4.1 La función DIMN

El SuperBASIC dispone de una función incorporada, la DIMN, que devuelve el tamaño máximo de una de las dimensiones especificadas de la matriz. Si la dimensión no se menciona en la instrucción, el intérprete de SuperBASIC asumirá la primera de ellas.

El formato de utilización de esta instrucción es:

DIMN (*matriz* [, *dimensión*])

donde *matriz:* es el nombre de la matriz previamente declarada en la función.

dimensión: representa el número de la dimensión a la que se hace referencia.

Si la *dimensión* especificada en la propia instrucción no existe o bien la *matriz* no está definida como tal, entonces la función DIMN devolvera el valor 0 (cero).

Ejemplos

Sea la declaración de matriz:

```
10   DIM m(3, 4, 5)
```

Estudiemos los siguientes ejemplos:

```
20   PRINT DIMN(m, 1)
imprimira 3

30   PRINT DIMN(m, 2)
imprimira 4

40   PRINT DIMN(m)
imprimirá 3 (la 1ª dimensión por defecto)

50   PRINT DIMN(m, 4)
imprimirá 0. (no existe la cuarta dimensión)
```

4.5 MATRICES DE CADENAS

Las matrices que hemos visto en los ejemplos hasta ahora estudiados corresponden al grupo de matrices numéricas, es decir, matrices en cuyos elementos solamente podía haber números.

El SuperBASIC del QL también proporciona la facilidad de declarar y manejar matrices cuyos elementos sean cadenas de caracteres.

La forma de declaración de estas matrices es similar a la vista con anterioridad utilizando la sentencia DIMensión, pero esta vez el identificador que da nombre a la matriz debe terminar con el signo $ tal y como apuntábamos en el capítulo 2 para las reglas de formación de los identificadores.

Así, por ejemplo, la sentencia:

```
10   DIMension nombre$(3)
```

declara una matriz de cuatro elementos (numerados del cero al tres) y que cada uno de los cuales puede contener *una sola letra* o carácter que será tomado como un string.

Así pues, si quisiéramos disponer de una matriz que contuviera las cinco primeras letras del alfabeto podríamos escribir:

```
10   DIMension letra$(4)
20   letra$ = {"A", "B", "C", "D", "E" }
```

de forma que si a continuación escribiéramos:

```
30   PRINT letra$(0)
40   PRINT letra$(1)
50   PRINT letra$(2)
60   PRINT letra$(3)
70   PRINT letra$(4)
```

La ejecución de este segmento de programa daría como resultado:

```
A
B
C
D
E
```

Para el almacenamiento de palabras completas, deberemos declarar en la matriz de string una segunda dimensión que indicará la longitud de la máxima cadena que queremos almacenar.

Sea, por ejemplo, el programa siguiente:

```
10   DIMension nom$(2, 7)
20   nom$ = {"CARLOS", "GALAN", "PASCUAL" }
30   PRINT nom$(0)
40   PRINT nom$(1)
50   PRINT nom$(2)
```

Que dará como resultado:

```
CARLOS
GALAN
PASCUAL
```

Obsérvese que la segunda dimensión de la matriz nom$ anterior es un 7 dado que la cadena más larga que se quería almacenar es "PASCUAL".

Análogamente pueden construirse matrices de cadenas de dos dimensiones.

Sea, por ejemplo, la declaración de matriz:

```
10   DIMension nom$(2, 5, 8)
```

que poseerá la estructura mostrada en la figura 4.5.

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)

Fig. 4.5.— Estructura de matriz de dos dimensiones.

Donde cada una de las celdas podrá albergar una cadena de hasta 8 caracteres tal y como se define en la declaración anterior.

4.6 FUNCIONES DE CODIFICACION-DECODIFICACION

El SuperBASIC del QL posee dos funciones incorporadas que son las CHR$ y CODE que se utilizan para convertir los caracteres ASCII en su código numérico y viceversa.

La función CHR$ aplicada a un número entero devuelve el símbolo asociado con dicho número.

El formato general de esta función es:

CHR$ *(expresión-numérica)*

Así, la instrucción de programa:

```
10   PRINT CHR$(42)
```

imprimirá el carácter "*" dado que es éste precisamente el que corresponde al número decimal 42 en la tabla de símbolos.

La figura 4.6 muestra la tabla de caracteres ASCII más utilizados con sus correspondientes codificaciones decimales.

Símbolo	Código	Símbolo	Código	Símbolo	Código	Símbolo	Código
	32	!	33	"	34	#	35
$	36		37	&	38	'	39
(	40	)	41	*	42	+	43
,	44	—	45	.	46	/	47
0	48	1	49	2	50	3	51
4	52	5	53	6	54	7	55
8	56	9	57	:	58	;	59
<	60	=	61	>	62	?	63
@	64	A	65	B	66	C	67
D	68	E	69	F	70	G	71
H	72	I	73	J	74	K	75
L	76	M	77	N	78	O	79
P	80	Q	81	R	82	S	83
T	84	U	85	V	86	W	87
X	88	Y	89	Z	90	[	91
\	92	]	93	^	94	_	95
£	96	a	97	b	98	c	99
d	100	e	101	f	102	g	103
h	104	i	105	j	106	k	107
l	108	m	109	n	110	o	111
p	112	q	113	r	114	s	115
t	116	u	117	v	118	w	119
x	120	y	121	z	122	{	123
:	124	}	125	à	139	è	143
	147	ò	150	ù	153	ñ	136
Ñ	168						

Fig. 4.6.– Tipos y códigos ASCII más usuales en castellano.

Veremos ahora la función inversa a la CHR$.

La función CODE aplicada a un carácter de string devuelve el número entero representativo de tal carácter según la tabla de la figura 4.6.

El formato más general de esta función es:

CODE *(expresión-cadena)*

donde *expresión-cadena:* puede ser, en el caso más general, un carácter de string, una tira de caracteres o una expresión entre tiras de caracteres relacionados con los operadores adecuados.

Así, la instrucción de programa:

```
10   PRINT CODE "A"
```

imprimirá el valor 65 representativo del carácter "A" mayúscula.

Cuando se aplica la función CODE a un string de más de un símbolo, sólo se ve afectado el primero de ellos.

Así pues, la instrucción:

```
10   PRINT CODE "GALAN"
```

devolverá el valor 71 representativo de la letra "G" mayúscula.

El argumento de la función CODE ha de ser, como hemos dicho, de tipo string explícitamente, como en el caso anterior, o implícitamente utilizando una variable string como en el caso:

```
10   a$ = "GALAN"
20   PRINT CODE a$
```

En este último caso ya no es necesario entrecomillar a$ en la línea 20 dado que se trata de una variable de tipo string propiamente dicha.

4.7 FRAGMENTACION DE CADENAS (SLICING)

El SuperBASIC del QL posee la facilidad del tratamiento fraccionado o particionado de cadenas de forma que se tenga acceso a sólo una parte de la cadena según qué reglas se utilicen para su denotación.

No solamente pueden fraccionarse cadenas de caracteres sino también matrices de cadenas e incluso matrices numéricas.

Se trata, por ejemplo, de imprimir ciertos segmentos de una cadena. Podemos hacer:

```
10    a$ = "SUPERBASIC QL"
20    PRINT a$
```

En esta caso, la salida sería la cadena a$ completa:

```
SUPERBASIC QL
```

que consta de 13 caracteres incluyendo el blanco separador.

Para la impresión del primer carácter exclusivamente de una cadena bastaría con añadir un subíndice al nombre de la cadena.

Esto es

```
30    PRINT a$(1)
```

obtendría como salida el carácter "S".

Para la referenciación de una subcadena dentro de otra cadena mayor basta con mencionar entre qué dos caracteres queremos que se fragmente la cadena, separados por la palabra TO.

Así, la instrucción:

```
40    PRINT a$(2 TO 5)
```

para la misma cadena anterior, dará como resultado:

```
UPER
```

Cuando se pretende referenciar a una subcadena hasta el final y a partir de un determinado elemento se suprime el último de los números.

La instrucción

```
50    PRINT a$(5 TO)
```

dará como resultado:

```
RBASIC QL
```


Obsérvese que tanto en la línea de programa 40 como en la 50, los números 2 y 5 que hacen referencia a los símbolos de la cadena ocupan la segunda y la quinta posición respectivamente.

Cuando se pretende referenciar a una subcadena dentro de una cadena que la contiene hasta una posición determinada se omite el primer número delante de la palabra TO.

La ejecución de la instrucción:

```
60    PRINT a$(TO 7)
```

dará como resultado:

```
SUPERBA
```

Esta forma de fragmentación (slicing) permite el tratamiento de subcadenas o partes de cadenas incluidas en una cadena completa.

El formato siguiente muestra todas las formas de escritura utilizando fragmentación:

(símbolo-1 **TO** símbolo-2)	desde símbolo-1 hasta símbolo-2
(símbolo-1 **TO**)	desde símbolo-1 hasta el final
(**TO** símbolo-2)	desde el principio hasta símbolo-2

En SuperBASIC pueden utilizarse conjuntamente las facilidades de conversión (coerción) y fragmentación (slicing) de cadenas.

El siguiente programa muestra un ejemplo de lo dicho:

```
10    nota$ = "987654"
20    nota$(3 TO 4) = 3 + nota$(3 TO 4)
30    PRINT nota$
```

que dará como resultado:

```
987954
```

4.8 INCLUSION DE CADENAS: EL OPERADOR INSTR

El SuperBASIC dispone del operador INSTR que determina si una subcadena dada se encuentra incluida dentro de otra cadena.

El formato de su utilización es:

cadena-1 **INSTR** *cadena-2*

Si la *cadena-1* está contenida en *cadena-2* entonces el operador INSTR devuelve la *posición de comienzo* de la cadena-1 dentro de la cadena-2. Si no existiera, devolverá un *cero*.

Ejemplos

```
10   PRINT "a" INSTR "Carlos"
imprimira 2

10   PRINT "basic" INSTR "superbasic"
imprimira 6

10   PRINT "x" INSTR "Isabel"
imprimira 0
```

Naturalmente, el operador INSTR debe ser utilizado en el contexto de cualquier instrucción adecuada (p. ej. IF, SELect, etc.).

Así, por ejemplo, puede ser utilizado dentro de una instrucción IF de forma que el desarrollo de la evaluación de la condición será TRUE si la cadena está incluida y FALSE en caso contrario.

```
40   IF cad1$ INSTR cad2$ THEN . . .
```

No obstante, en el capítulo 5 se estudiarán con más detalle estas instrucciones condicionales.

4.9 REPETICION DE CADENAS: LA FUNCION FILL$

Cuando se desea repetir un carácter o varios un número de posiciones elevado puede utilizarse la instrucción FILL$.

El formato general de esta función es:

FILL$ *(expresión-cadena, expresión-numérica)*

donde *expresión-cadena:* puede ser o bien una constante de string, una variable de cadena o incluso una expresión de string.

expresión-numérica: puede ser o bien una constante numérica, una variable o incluso una expresión de este tipo.

Ejemplos

```
10   PRINT FILL$("=", 7)
imprimirá = = = = = = =

40   PRINT FILL$("hola", 4)
imprimirá  holaholaholahola
```

4.10 LA FRAGMENTACION EN MATRICES

Acabamos de estudiar como utilizar la característica de la fragmentación en cadenas simples. No obstante, estos mismos conocimientos pueden aplicarse a las matrices de cadenas *(string array)*.

Observe el lector el siguiente programa y trate de decidir cuales serían los resultados obtenidos con los conocimientos que ya posee.

```
10   DIMension nom$(3, 7)
20   nom$(0) = "carlos"
30   nom$(1) = "galan"
40   nom$(2) = "isabel"
50   nom$(3) = "cordero"
60   PRINT nom$(0) (1 TO 3)
70   PRINT nom$(1) (TO 4)
80   PRINT nom$(2) (3 TO)
90   PRINT nom$(3) (4)
```

En el programa anterior se ha declarado en la línea 10 una matriz de cadenas llamada nom$ que poseerá 4 cadenas (accesibles de la 0 a la 3) y con una longitud máxima para cada una de ellas de 7 caracteres.

En las líneas 20 a 50 se procede a llenar la matriz anterior con las cadenas que se mencionan.

En la línea 60 se hace referencia al primer elemento de la matriz, llamado NOM$(0) que contiene en este momento la cadena "carlos". En esta misma línea se está diciendo que de la cadena completa "carlos" se impriman exclusivamente los caracteres comprendidos entre el primero y el tercero (1 TO 3). De esta forma se obtendrá como resultado:

```
car
```

En la línea 70 se hace referencia al segundo elemento de la matriz, llamado NOM$(1) que contiene en este momento la cadena "galan". Con la expresión de fragmentación (TO 4) se está indicando que se impriman exclusivamente los caracteres desde el comienzo de la cadena hasta el cuarto. La salida será, por tanto:

```
gala
```

En la línea 80 se hace referencia al tercer elemento de la matriz, llamado NOM$(2) que contiene en este momento la cadena "isabel". Con la expresión de fragmentación (3 TO) se está indicando que se impriman exclusivamente los caracteres desde el tercero hasta el final de la cadena. La salida será:

```
abel
```

En la línea 90 se hace referencia al cuarto elemento de la matriz, llamado NOM$(3) que contiene en este momento la cadena "cordero". Con la expresión (4) que sigue a este elemento se está indicando que se imprima exclusivamente el carácter cuarto. La salida, por tanto, sera:

```
d
```

Note el lector que las instrucciones:

```
100  PRINT nom$(1) (TO 1)
110  PRINT nom$(1) (1)
```

son equivalentes y producen el mismo resultado: "g".

De forma análoga puede utilizarse la característica de la fragmentación de cadenas en la asignación de manera que una cadena fragmentada pueda ser objeto o sujeto de una instrucción de este tipo.

Obsérvese la instrucción siguiente:

```
b$ = "sinclair ql" (3 TO 6)
```

Al final de la ejecución de la misma, la variable-string b$ contendrá aquel fragmento de la cadena "sinclair ql" comprendido entre los caracteres tercero al sexto, es decir, contendrá "ncla".

Aquí acabamos de ver cómo la fragmentación ha sido utilizada en la parte derecha del igual en la sentencia de asignación. Igualmente puede utilizarse en la parte izquierda de la instrucción de forma que sólo se vea afectado un fragmento de la cadena.

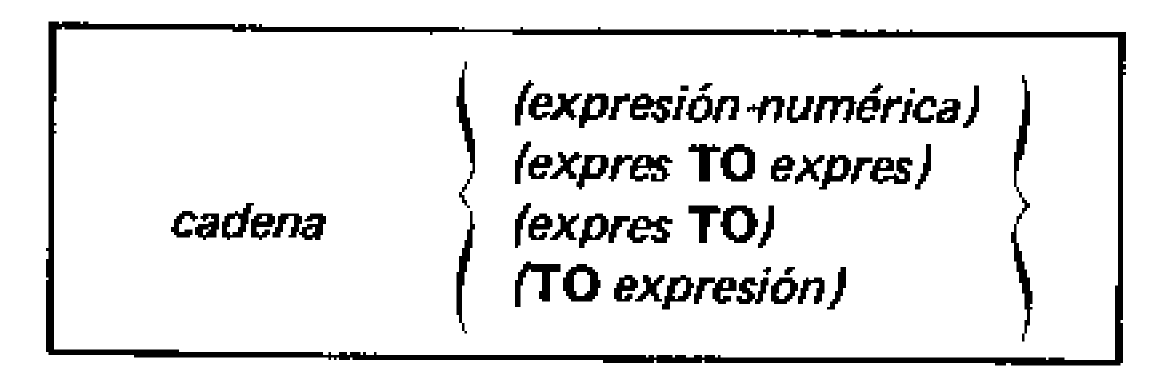
Sea, por ejemplo, el fragmento de programa siguiente:

```
10   cadena$ = "programacion"
20   cadena$(4 TO 8) = "basic"
30   PRINT cadena$
```

En la línea 10 se inicializa la cadena$ con "programación" y en la línea 20 se asigna la cadena "basic" al fragmento de cadena$ comprendido entre los caracteres cuarto al octavo, de forma que la salida será:

```
probasiccion
```

Con esto hemos querido demostrar como allí donde aparezca una cadena, esta siempre puede ir acompañada de las características de fragmentación.



Su utilización no está, por tanto, limitada.

Sea el siguiente programa:

```
10   DIMension nom$(2, 4)
20   nom$(0) = "carlos"
30   nom$(1) = "isabel"
40   nom$(2) = "blas"
50   nom$(0) (1 TO 2) = nom$(1) (TO 2)
60   nom$(1) = nom$(1) (4) & nom$(0)
```

```
70   PRINT nom$(0)
80   PRINT nom$(1)
```

En la línea 10 se ha declarado una matriz de 3 cadenas (accesibles del 0 al 2) con una longitud máxima para cada una de ellas de 4 caracteres.

En la línea 20 se asigna o llena el primer componente de la matriz nom$(0) con la cadena "carlos". Dado que la longitud máxima de cada cadena en la matriz es de 4 elementos, el SuperBASIC truncará dicha cadena, de forma que nom$(0) contendrá exclusivamente la cadena "carl". Lo mismo sucede en la línea 30 con el elemento nom$(1) que contendrá "isab".

La asignación de la línea 40 permite que la cadena entera "blas" se introduzca en nom$(2).

En la línea 50 se asignan a los tres primeros caracteres de nom$(0) los dos primeros caracteres de nom$(1) de forma que en este momento

nom$(0) contiene "isrl"

En la línea 60 se concatenan dos cadenas, esto es; se concatenan el cuarto elemento de nom$(1) con todos los elementos de nom$(0) de forma que en este momento

nom$(1) contiene "bisr"

Obsérvese en este último caso que se ha producido el truncado del carácter "1" al hacer la concatenación de ambas cadenas por la condición impuesta en la línea 10 limitando a 4 el número máximo de caracteres en cada elemento de la matriz.

Análogamente puede hablarse de fragmentación para matrices estrictamente numéricas.

Sea, por ejemplo, el programa siguiente:

```
10   DIMension a(2, 2)
20   DIMension b(3, 3)
30   b(0, 0) = 0.0: b(0, 1) = 0.1
40   b(0, 2) = 0.2: b(0, 3) = 0.3
50   b(1, 0) = 1.0: b(1, 1) = 1.1
60   b(1, 2) = 1.2: b(1, 3) = 1.3
70   b(2, 0) = 2.0: b(2, 1) = 2.1
80   b(2, 2) = 2.2: b(2, 3) = 2.3
90   a = b(0 TO 2, 0 TO 2)
```


La disposición de las matrices A y B al final de la ejecución del programa anterior se muestra en la figura 4.7.

MATRIZ A

	0	1	2
0	0.0	0.1	0.2
1	1.0	1.1	1.2
2	2.0	2.1	2.2

MATRIZ B

	0	1	2	3
0	0.0	0.1	0.2	0.3
1	1.0	1.1	1.2	1.3
2	2.0	2.1	2.2	2.3
3	0	0	0	0

Fig. 4.7.– Ejemplo de fragmentación para matrices numéricas.

Como puede observarse en la figura 4.7, la instrucción:

```
90   a = b (0 TO 2, 0 TO 2)
```

procede a llenar la matriz A con el fragmento de la matriz B que se encuentra definido por los paréntesis.

A diferencia de la fragmentación para cadenas o matrices de cadenas, la fragmentación para matrices numéricas debe poseer la característica de ser compatibles en cuanto a tamaño la parte izquierda del signo igual con la parte derecha, tal y como se muestra en la línea 90 del programa anterior.

4.11 LA LONGITUD DE LAS CADENAS: LA FUNCION LEN

El SuperBASIC del QL dispone de una función llamada LEN que aplicada a una cadena o a una variable de tipo cadena devuelve la *longitud* de dicho string.

El formato general de la función LEN es:

LEN *(expresión-cadena)*

donde *expresión-cadena:* puede ser o bien una constante o una variable string o incluso una expresión de string que contenga los operadores adecuados.

Sea, por ejemplo, el siguiente programa:

```
10   INPUT "escriba su nombre"; nom$
20   PRINT
30   PRINT "su nombre es, "; nom$;
40   PRINT " y tiene "; LEN(nom$);
50   PRINT " caracteres de longitud"
```

Si, por ejemplo, en la línea 10, escribiéramos "carlos", la salida de este programa sería:

su nombre es, carlos y tiene 6 caracteres de longitud

En este caso se ha aplicado la función LEN a una variable de tipo string.

También podría aplicarse directamente a una cadena:

```
10   PRINT LEN("sinclair ql")
```

que obtendría como salida: 11

Recordemos ahora el manejo y la utilización de la concatenación de cadenas. Observemos esto con el programa siguiente.

Se recomienda al lector su estudio detallado antes de proceder a introducirlo en su ordenador (1).

```
10   REPeat letras
20     prim$ = CHR$(RND(65 TO 67) )
30     seg$ = CHR$(RND(65 TO 67) )
40     ter$ = CHR$(RND(65 TO 67) )
50     palabra$ = prim$ & seg$ & ter$
60     PRINT !palabra$!
70     IF palabra$ = "ABC" THEN EXIT letras
80   END REPeat letras
```

Obsérvese que en las instrucciones de las líneas 20, 30 y 40 se obtiene un carácter en cada una que después se concatenan juntos para formar palabra$.

(1) Nota: Consulte el lector, si lo cree necesario, el uso de la función de generación de números aleatorios RND en el capítulo 8. La instrucción REPeat y la sentencia IF serán estudiadas con detalle en los siguientes apartados.

5

Instrucciones alternativas y de bifurcación

5.1. INTRODUCCION

Una de las características más importantes del lenguaje SuperBASIC para el QL es la posibilidad de incluir sentencias de programación estructurada, que lo acercan cada vez más a lenguajes del tipo PASCAL, C, etc., Así, de esta forma, y con la utilización de estas sentencias nuevas que veremos con detalle en el presente capítulo y en el siguiente, podremos muy bien omitir las tradicionales instrucciones GOTO, GOSUB, etc, que contribuyen a hacer menos legible el programa y menos estructurado.

No obstante, y con el fin de compatibilizar el SuperBASIC con otros BASIC's anteriores, el repertorio de instrucciones tradicionales siempre puede usarse aunque, como veremos, no es recomendable.

Las ventajas de la utilización del SuperBASIC estructurado se ponen de manifiesto especialmente en el control del programa, en los bucles controlados, en las decisiones alternativas y en las bifurcaciones.

En este capítulo se estudiarán con detalle las instrucciones de salto GOTO (bifurcación incondicional), ON ... GOTO (bifurcación condicional), la sentencia IF...THEN...ELSE (de prueba de condición para la ejecución exclusiva de una de dos tareas propuestas) y sus correspondientes variantes con el uso de los operadores lógicos OR, AND, XOR y NOT; la sentencia SELECT (de prueba de condición para la ejecución de una de varias tareas propuestas) y las instrucciones PAUSE y STOP de parada temporal y definitiva respectivamente de un programa.

Con todas ellas se cubre todo el posible espectro de necesidades en el terreno de la escritura de programas que posean saltos condicionales o incondicionales y selección entre varias alternativas posibles.

Utilizaremos, siempre que ello sea factible a nivel de claridad de comprensión y de estudio, las técnicas de programación estructurada, que ya pueden ser utilizadas en este SuperBASIC.

5.2. BIFURCACION INCONDICIONAL: LA INSTRUCCION GOTO

El tipo de instrucción de *bifurcación incondicional* por excelencia en casi todos los lenguajes de programación es la sentencia GOTO, que posee el formato:

GOTO *línea*

donde *línea:* es un número de línea de cualquiera de las que esté formado el programa.

Ejemplo:

```
210   GOTO 115
```

Cuando se ejecuta la instrucción anterior, el programa bifurca incondicionalmente al número de línea señalado después de la palabra GOTO (en este caso, a la instrucción de la línea 115).

El número de la línea a la que se salta puede ser mayor o menor que la línea actual de forma que esta bifurcación puede realizarse bien hacia adelante o bien hacia atrás en un programa.

El segmento de programa siguiente:

```
10   PRINT "bucle infinito"
20   GOTO 10
```

visualizaría infinitas veces la cadena de la línea 10, a no ser que parasemos la ejecución del programa con un *break* (CTRL y espacio).

5.3. BIFURCACION CONDICIONAL: La instrucción ON...GOTO

La sentencia ON...GOTO se utiliza para transferir el control del programa a una de varias líneas del mismo, dependiendo del valor de una expresión que en el caso más sencillo puede ser una variable exclusivamente. La instrucción ON tiene dos partes: una variable y un control de bifurcación.

El formato general de una sentencia ON...GOTO es:

ON *variable* **GOTO** *expresión* [{*,expresión*} ^{n}]

donde *variable:* debe ser un identificador.

expresión: puede ser o bien una constante o variable numérica o incluso una expresión de tal tipo.

El control de bifurcación posee la palabra GOTO seguida por dos o más números de líneas separados por comas.

Sea una posible forma de la instrucción ON ... GOTO:

ON variable **GOTO** ℓ_1, ℓ_2, ..., ℓ_n

Si el valor de la variable es 1, el control se transfiere a la primera línea referenciada (ℓ_1) después de la palabra GOTO; si la variable vale 2, el control se transfiere a la segunda línea escrita (ℓ_2), etc.

Sea el siguiente programa:

```
10   INPUT "teclear un nro.", a
20   ON a GOTO 40,60,80
30   GOTO 10
40   PRINT "numero=1"
50   GOTO 10
60   PRINT "numero=2"
70   GOTO 10
80   PRINT "numero=3"
90   GOTO 10
```

Si el valor leído de "a" es un 1, el control del programa se transferirá a la línea 40; si el valor es 2, el control será para la línea 60, etc.

Si el valor de la *variable* de una instrucción ON...GOTO es menor que 1 ó mayor que el número de líneas escritas en la propia instrucción, entonces se producirá un error de ejecución del programa.

Como ya veremos, las instrucciones GOTO y ON...GOTO no son necesarias en SuperBASIC, pues se dispone de otras instrucciones de tipo estructurado que no hacen recomendable su uso supliéndolas con gran ventaja.

5.4. PRUEBAS DE CONDICION: La instrucción IF

Esta instrucción plantea una o dos salidas posibles a una condición impuesta en la propia sentencia.

El formato más sencillo de la instrucción es:

```
IF condición THEN instrucción
```

Si el resultado de la evaluación de la condición resulta ser *verdadero (true)* entonces se pasarán a ejecutar las instrucciones escritas detrás de la palabra THEN *(bloque-THEN).*

Algunos ejemplos de su utilización puede ser:

```
10   IF micros$ = "QL" THEN PRINT "Sinclair"
20   IF a + b < c * d THEN GOTO 60
```

Este es el caso más simple. Una instrucción IF posee dos partes claramente diferenciadas: una *condición* y una *acción* a realizar si dicha condición resulta ser cierta.

En la instrucción de la línea 10 anterior la condición impuesta pregunta si el contenido de la variable-string MICROS$ coincide con la cadena "QL", si esto sucede así, entonces se procede a visualizar la cadena "Sinclair"; en caso contrario, esto es; cuando la condición no se cumple (MICRO$ contiene cualquier otra cosa) se pasa a ejecutar la siguiente instrucción en secuencia.

En la línea 20, la condición se ha expresado como una relación entre expresiones de tipo aritmético, así, si la expresión A + B es menor que C * D entonces se derivará el control del programa a la línea 60 que se menciona en la propia instrucción.

Cuando se comparan cadenas en una relación, esta comparación se realiza alfabéticamente. Por eso la relación "a"<"b" es verdad (true) porque "a" se encuentra delante de "b" alfabéticamente. Cuando se comparan cadenas de diferentes longitudes, la cadena más corta se amplía con blancos durante la comparación con el fin de hacerlas de igual longitud. Por eso "a"<"aa" es verdad dado que la cadena "a" será tratada como "a " para los efectos de la comparación y el blanco tiene alfanuméricamente menor valor que la letra "a".

En los casos anteriores, después de la palabra THEN sólo había una instrucción a ejecutar. Cuando se desea que se ejecuten varias instrucciones siempre que se cumpla la condición impuesta en la sentencia IF, entonces se hace necesario añadir las palabras END IF al final de la última de las instrucciones mencionadas.

El formato que define esto podría expresarse como:

```
IF condición THEN
                 sentencia-1
                 sentencia-2
                 ................
                 sentencia-n

END IF
```

Sea, por ejemplo, el siguiente programa:

```
10   IF micro$ = "ql"  THEN
20               PRINT "Sinclair"
30               PRINT "SuperBASIC"
40               ql = ql + 1
50   END IF
```

En el programa anterior, cuando se da la circunstancia de que la variable-string vale "ql" entonces (THEN) se pasa a ejecutar la instrucción o grupo de instrucciones comprendidas entre las palabras THEN... END IF.

Si, por ejemplo, el resultado de la evaluación de la condición resulta ser falsa, entonces se saltan todas las instrucciones hasta localizar END IF y se pasa a ejecutar la siguiente instrucción en secuencia.

Existe una versión más general de la instrucción IF de la forma:

IF *condición* **THEN** *instrucciones-1* **ELSE** *instrucciones-2*

Cuando el programa SuperBASIC se encuentra con una instrucción de este tipo, lo primero que hace es evaluar la *condición*. Si esta condición resulta ser *verdadera (true)* se ejecutan las *instrucciones-1 (bloque-THEN)*. Cuando, por el contrario, la condición no se cumple, es decir, es *falsa (false)*, el programa ejecutará las *instrucciones-2 (bloque-ELSE)*. En cualquiera de ambos casos, cuando se termina la ejecución del bloque-THEN o del bloque-ELSE, se deriva el control a la siguiente instrucción después de la IF.

Con este nuevo formato de la instrucción se da la oportunidad al QL de que escoja entre dos posibles alternativas.

Sea el siguiente programa ejemplo:

```
10   IF micro$ = "QL" THEN
20      PRINT "SuperBASIC"
30   ELSE
40      PRINT "BASIC normal"
50   END IF
60   a = a + 1
```

En el programa anterior, cuando la variable-string MICRO$ contiene la cadena "QL" entonces (THEN) se procede a imprimir la cadena "SuperBASIC". En caso contrario (ELSE), es decir, cuando MICRO$ contiene cualquier cadena distinta de "QL" se pasa a ejecutar el bloque-ELSE y se imprimirá la cadena "BASIC normal". En cualquiera de los dos casos después de ejecutar la instrucción PRINT correspondiente se derivará el control a la instrucción siguiente a la delimitadora END IF, en nuestro caso, a la sentencia de asignación de la línea 60.

El SuperBASIC del QL permite que la palabra THEN pueda ser omitida dentro del contexto de una instrucción IF. El QL comprenderá perfectamente la instrucción siempre y cuando existan dos puntos (:) después de la condición.

Así, las dos instrucciones siguientes son equivalentes

```
10   IF micro$ = "QL" THEN PRINT "Sinclair"
```

o bien,

```
10   IF micro$ = "QL" : PRINT "Sinclair"
```


Pueden omitirse, no obstante, también los dos puntos (:) como en el ejemplo siguiente, cuando son varias las instrucciones que componen el bloque-THEN.

```
10   IF micro$ = "QL"
20             PRINT "Sinclair"
30             PRINT "BASIC ESTRUCTURADO"
40   END IF
```

Acabamos de ver como dentro de las instrucciones IF escribimos completamente las condiciones.

Sea, por ejemplo, la instrucción siguiente:

```
IF puntos = 1 THEN GOSUB 500
```

En ella se ha mencionado explícitamente la condición "puntos = 1" y en caso afirmativo se procedería a ejecutar la subrutina de la línea 500 (1).

Como ya veremos más adelante y en especial dentro del capítulo 8, una subrutina GOSUB puede ser perfectamente simulada con un *procedimiento (procedure)*.

Una condición puede ser escrita en SuperBASIC como una expresión o en el caso más sencillo como un identificador solo. Así, para la sentencia anterior, podría escribirse:

```
IF puntos THEN anotar
```

La condición en este caso viene determinada exclusivamente por el identificador "puntos". El SuperBASIC dará como resultado *TRUE* si el valor de la variable real "puntos" es *distinto de cero* y *FALSE* cuando sea *igual a cero*.

La palabra *anotar* que sigue a continuación puede ser perfectamente el nombre de un procedimiento (procedure) que es llamado cuando la condición se cumple.

A estas variables utilizadas como condiciones se las suele llamar *variables lógicas* y pueden utilizarse en SuperBASIC en conjunción con los operadores lógicos tradicionales NOT, AND, OR y XOR que se verán seguidamente.

(1) Nota: En el capítulo 7 se estudiarán con detalle las instrucciones para subrutinas.

Ejercicio

El siguiente programa lee un número y calcula y escribe su factorial. Como es sabido, el factorial de un número n es:

$$n! = n * (n-1) * (n-2) * \ldots * 2 * 1$$

```
 10  REMark ----------------------
 20  REMark     Calculo factorial
 30  REMark ----------------------
 40  INPUT "teclee un numero"!num
 50  i = 1
 60  fact = 1
 70  IF = num THEN
 80                      PRINT "factorial=";fact
 90              ELSE
100                      i = i + 1
110                      fact=fact*i
120                      GOTO 70
130  END IF
140  STOP
```

5.5. RELACIONES COMPUESTAS

Hemos visto hasta ahora como podíamos relacionar dos operandos según una serie de criterios (operadores de relación) que venían impuestos por la propia relación que se empleara en cada caso. Todo esto destinado a realizar una de dos posibles alternativas.

Una representación gráfica del uso de estas sentencias comparativas puede ser el de un conmutador eléctrico que se activara en una u otra posición dependiendo de que se cumpla o no la condición impuesta.

La figura 5-1 muestra un ejemplo de esto.

En el caso de la figura anterior, la condición impuesta por una instrucción IF se cumple, pasándose a ejecutar las instrucciones correspondientes al bloque-THEN.

En la teoría de circuitos suele representarse la condición IF por un *switch* o *interruptor* normal que se encuentra en posición de *conectado* o *desconectado* dependiendo de si se cumple o no la condición inmersa en la propia instrucción, respectivamente. En este caso podemos decir que no existe conducción eléctrica para el bloque-ELSE. La figura 5-2 muestra este esquema más simplificado.

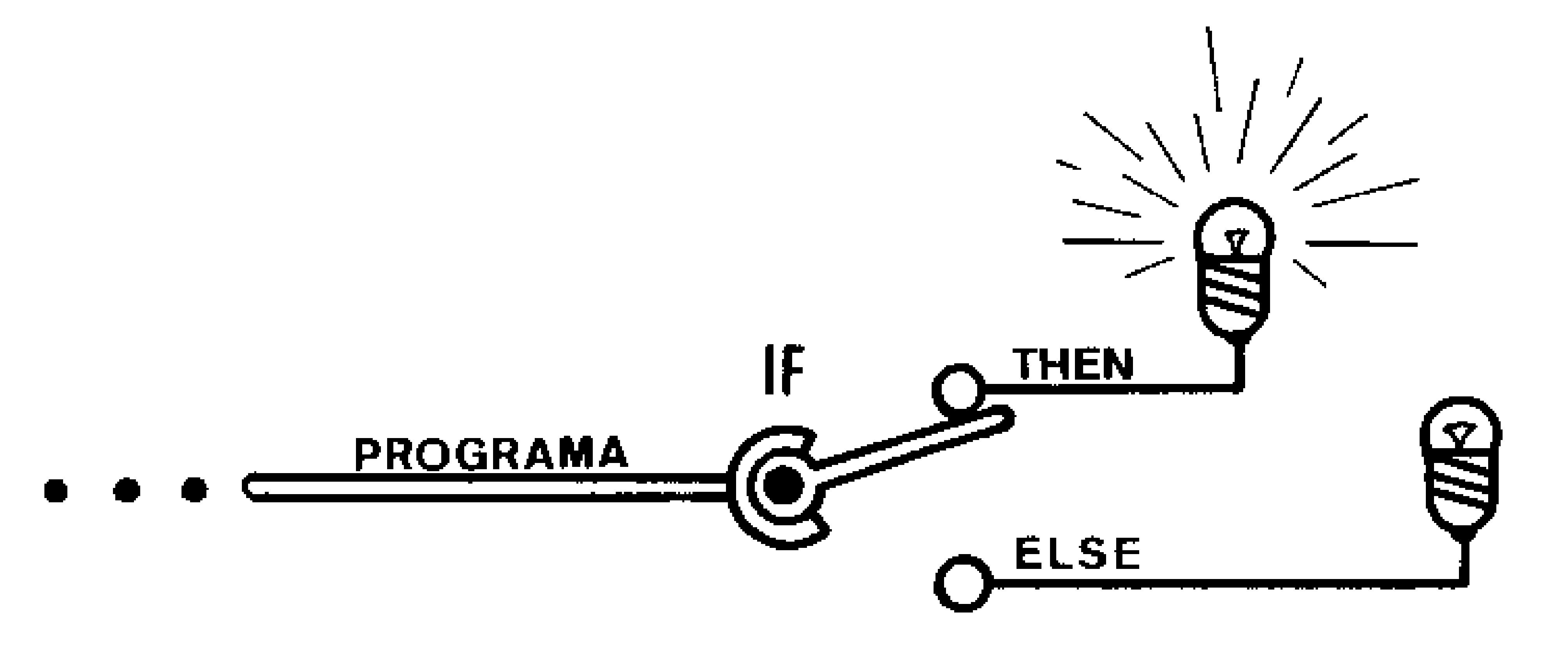


Fig. 5-1.— Ilustración de la sentencia IF.

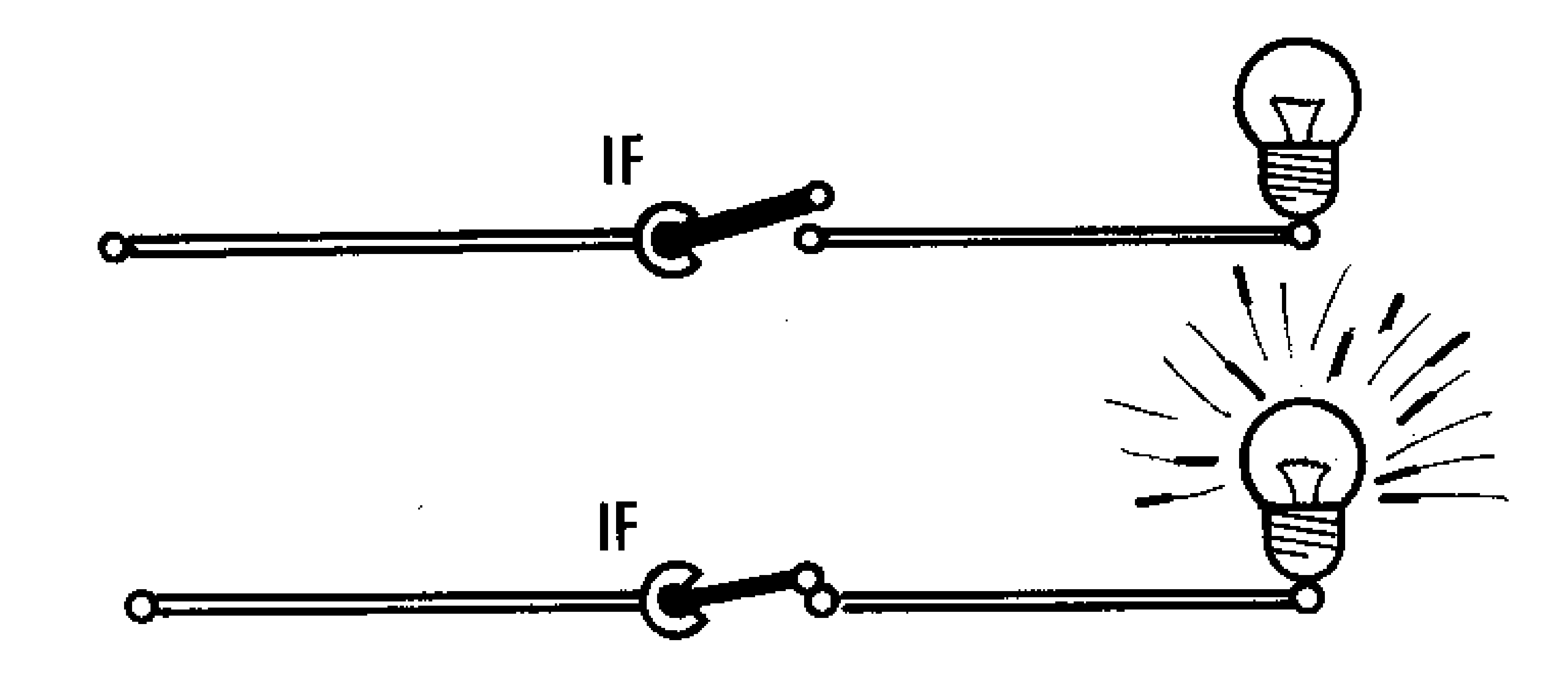


Fig. 5.2.— Circuito abierto y cerrado para la instrucción IF.

En este caso, el cumplimiento de la condición impuesta por la instrucción IF implica el cerramiento del circuito.

Todos los casos que hemos visto hasta ahora corresponden al tipo de *relación-simple*, es decir, para que se ejecutaran las instrucciones correspondientes al bloque-THEN bastaría con que se cumpliera la condición, *una única* condición. Sucede con frecuencia en programa-

ción práctica que se hace necesario conocer el resultado de varias condiciones antes de proceder a ejecutar una cosa u otra. A estas condiciones se les llama *relaciones-compuestas* y son aquellas que están formadas por dos o más relaciones simples. La forma de *composición* de relaciones puede obtenerse según los criterios descritos en los siguientes apartados.

5.5.1. El Operador Lógico OR

Cualquier condición, ya sea simple o compuesta, comporta, una vez efectuada, dos posibles salidas representadas universalmente como *TRUE (verdad:* se cumple la condición) y *FALSE (falsedad:* no se cumple la condición).

El operador lógico OR (al que podríamos traducir como: *cualquiera o ambas)* relaciona dos condiciones simples de modo que basta con que una sola de ellas (o las dos) se cumpla para que se cumpla la condición inmersa en la instrucción y se ejecute el bloque-THEN correspondiente.

Ejemplo:

Sea el organigrama de bloques de proceso mostrado en la figura 5-3.

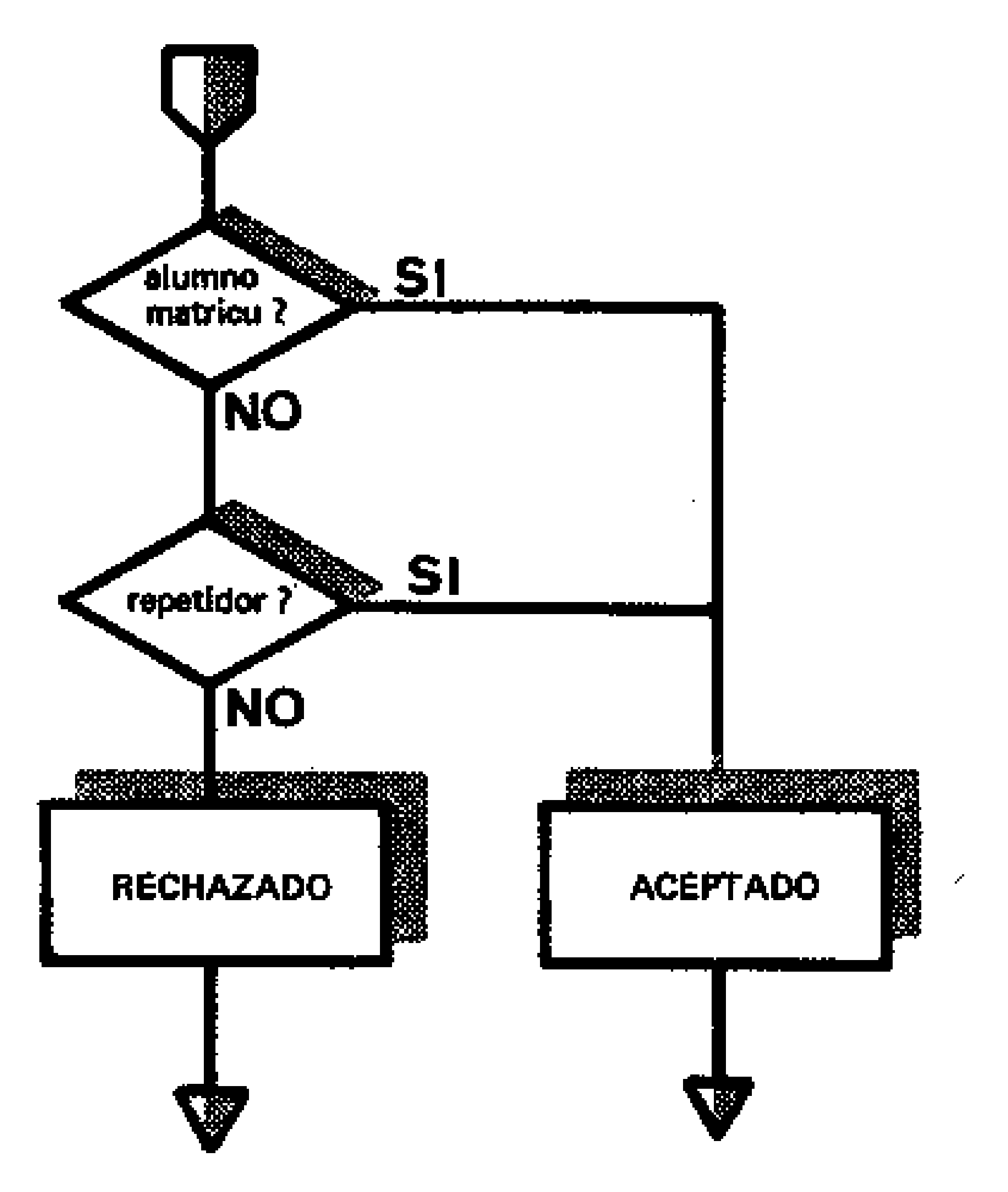


Fig. 5.3.– Estructura del operador lógico OR.

En este organigrama puede observarse como para que se ejecuten las instrucciones inmersas en el bloque-ACEPTADO basta con que se cumpla *una cualquiera* de las dos condiciones impuestas. Obvio resulta mencionar que si ambas condiciones se cumplen a priori, jamás llegará nuestro programa a evaluar la segunda de ellas.

Utilizando relaciones simples, el anterior organigrama podría ser codificado en SuperBASIC de la manera siguiente:

```
.
.
.
200  IF matricula$ = "si" THEN GOTO ... (aceptado)
210  IF repetidor$ = "si"  THEN GOTO ... (aceptado)
220  (rechazado)
.
.
.
```

Utilizando el operador lógico OR esta codificación podría ser simplificada del modo siguiente:

```
.
.
.
200  IF matrícula$ = "si" OR repetidor = "si" THEN
210  GOTO ... (aceptado) ELSE GOTO ... (rechazado)
.
.
.
```

En el ejemplo anterior, el programa bifurcará a las líneas de ACEPTADO cuando se cumpla la condición:

matrícula$ = "si" **OR** repetidor$ = "si"

Basta con que se cumpla una cualquiera de ellas (o las dos) para que esta relación-compuesta sea verdadera y se ejecuten las instrucciones correspondientes al bloque-THEN.

Cuando *ninguna* de las relaciones simples se cumple, el programa ejecutará la instrucción o instrucciones correspondientes al bloque-ELSE; en este caso la bifurcación a las líneas de RECHAZADO.

El cuadro de la figura 5-4 muestra la tabla de verdad respecto de la operación lógica OR con dos condiciones u operandos lógicos denominados C1 y C2.

C1	C2	C1 OR C2
SI	SI	SI
SI	NO	SI
NO	SI	SI
NO	NO	NO

Fig. 5.4.— Tabla de verdad de la relación lógica OR.

Las palabras SI o NO significan el cumplimiento o no de la condición.

Observando el cuadro de la figura 5-4 se desprende la conclusión de que el único caso donde *no se cumple* una condición compuesta OR es cuando *ambas* condiciones simples son falsas.

La representación gráfica de una condición compuesta del tipo OR mediante circuitos de interruptores sería colocando dichos interruptores *en paralelo* tal y como se muestra en la figura 5-5.

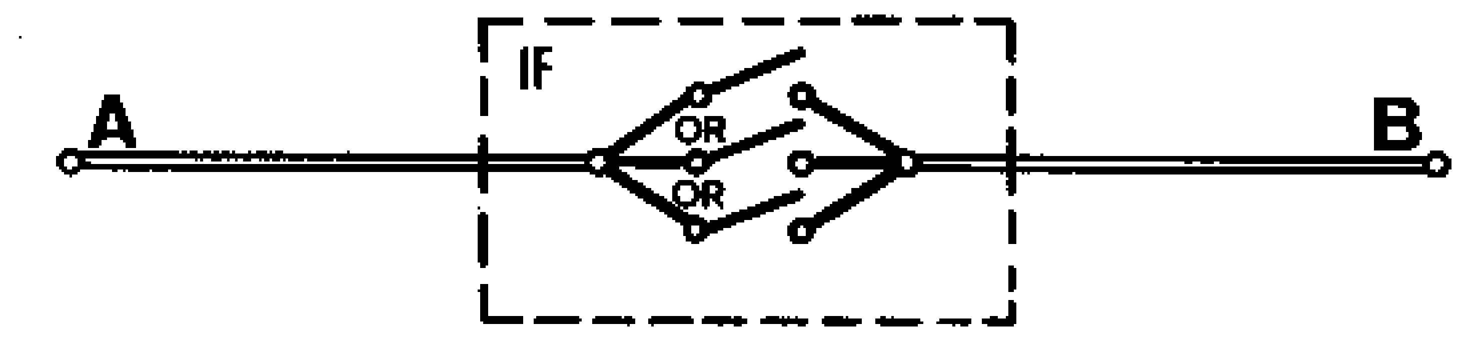


Fig. 5.5.— Circuitos en paralelo para la operación lógica OR.

En el caso de la figura 5-5, la instrucción IF está formada por tres condiciones unidas mediante dos operadores lógicos OR. Basta la observación de esta figura para comprobar lo anteriormente señalado; sólo se hace necesario que una cualquiera de las tres condiciones impuestas se cumpla para que llegue corriente del punto A al B.

5.5.2. El Operador Lógico AND

El operador lógico AND puede ser traducido por *y-además*. En este caso, *solamente* se ejecutará el bloque-THEN correspondiente a la instrucción IF cuando *todas* las condiciones impuestas se cumplan. En

caso contrario, el programa ejecutará las instrucciones correspondientes al bloque-ELSE.

Sea, por ejemplo, el diagrama de bloques de la figura 5-6.

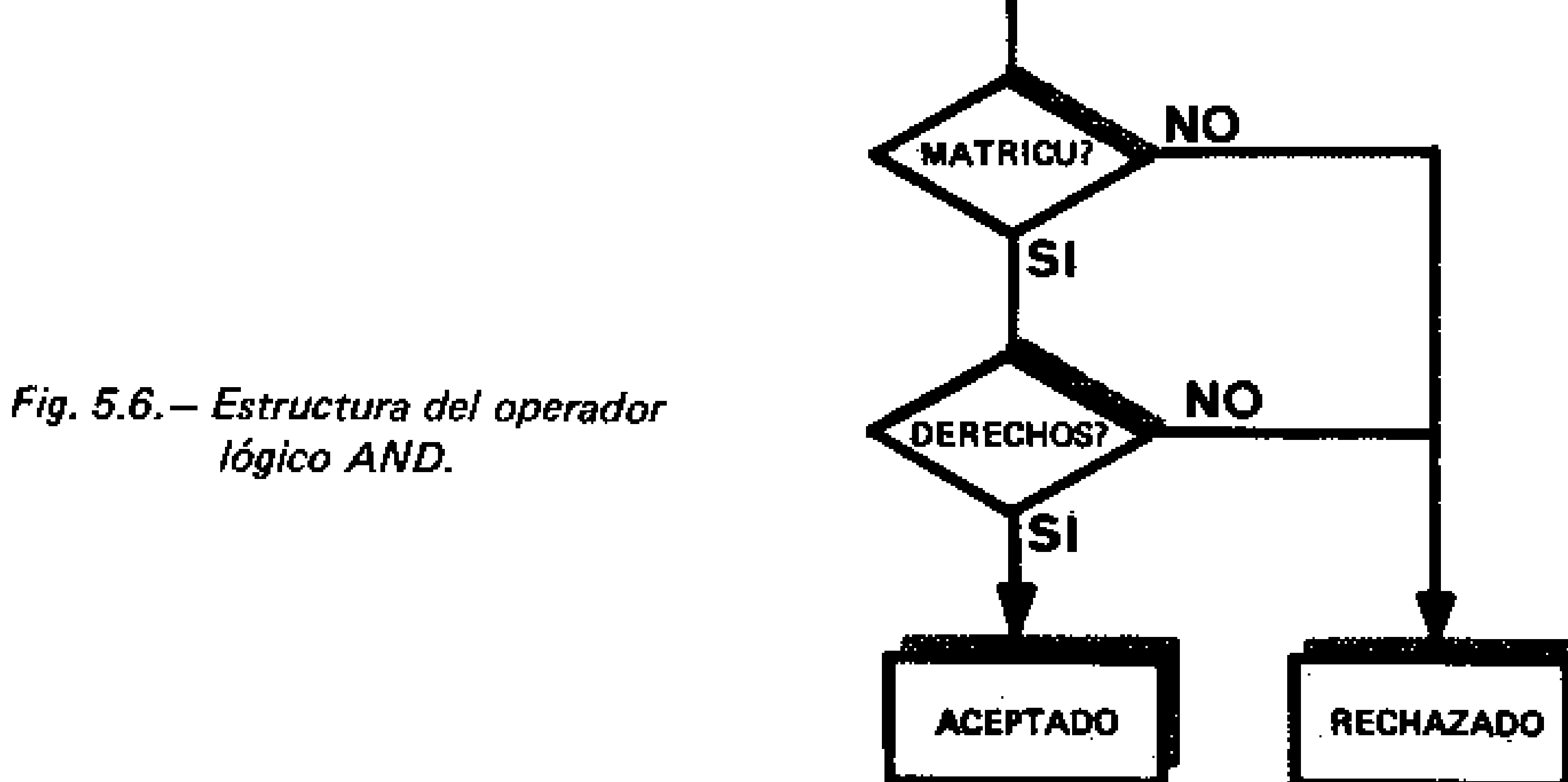


Fig. 5.6.— Estructura del operador lógico AND.

En este organigrama puede observarse como la necesidad de tener que cumplirse las dos condiciones impuestas es evidente si lo que se pretende es ejecutar las líneas de ACEPTADO. Basta con que una cualquiera (o las dos, por supuesto) de las condiciones no se cumpla, para que se ejecute el bloque-ELSE, en este caso, las líneas de RECHAZADO.

Con la utilización del operador lógico AND aplicado a la codificación de condiciones-compuestas, estas instrucciones serían equivalentes a la:

```
200  IF matrícula$ = "si" AND derechos$ = "si" THEN
210  GOTO ... (aceptado) ELSE GOTO ... (rechazado)
```

En esta condición, el programa bifurcará a las líneas de ACEPTADO si y sólo si se cumplen *todas* las condiciones inmersas en la instrucción, esto es; que los identificadores matrícula$ y derecho$ contengan la cadena "si".

El cuadro de la figura 5-7 muestra la tabla de verdad respecto de la operación lógica AND con dos condiciones u operandos lógicos denominados C1 y C2.

C1	C2	C1 AND C2
SI	SI	SI
SI	NO	NO
NO	SI	NO
NO	NO	NO

Fig. 5.7.— Tabla de verdad de la operación lógica AND.

En este caso, puede observarse como la condición compuesta por el ordenador AND es verdadera únicamente cuando lo son todas las que la componen.

La forma de representación gráfica de una condición compuesta del tipo AND mediante circuitos de interruptores sería colocando dichos interruptores *en serie* tal y como se muestra en la figura 5-8.

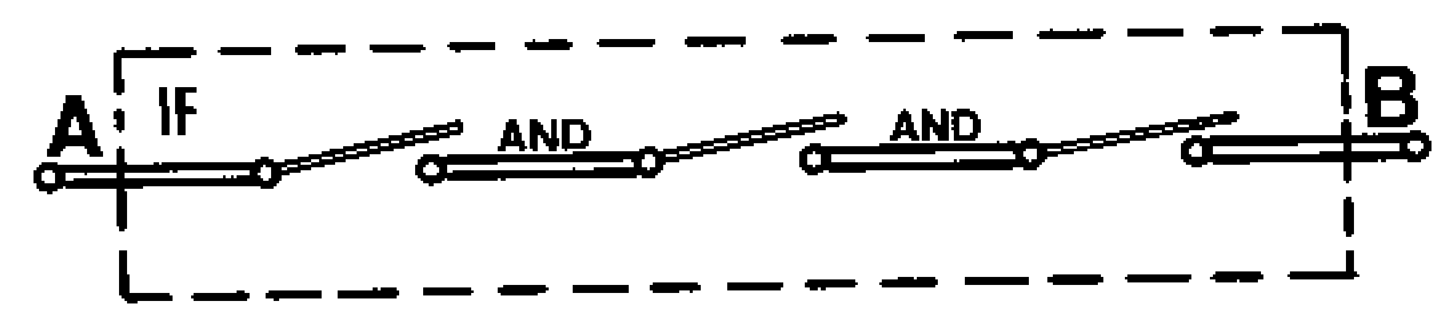


Fig. 5.8.— Circuitos en serie para la operación lógica AND.

En este caso, la instrucción IF está formada por tres condiciones unidas mediante dos operadores lógicos AND. Para que la corriente llegue al punto B partiendo del A, es absolutamente necesario que los tres interruptores representativos de las tres condiciones estén cerrados.

5.5.3. El Operador Lógico XOR

El operador lógico XOR también llamado *OR-exclusivo* puede traducirse por *"A o B pero no ambas"*. En este caso, se ejecutará el blo-

que-THEN cuando se cumpla *solamente una* de las dos posibles condiciones inmersas en una relación de este tipo. Cuando se cumplan las dos condiciones simultáneamente, entonces se pasa a ejecutar el bloque-ELSE.

El cuadro de la figura 5-9 muestra la tabla de verdad respecto de la operación lógica XOR con dos condiciones u operandos lógicos denominados C1 y C2.

C1	C2	C1 XOR C2
SI	SI	NO
SI	NO	SI
NO	SI	SI
NO	NO	NO

Fig. 5.9.— Tabla de verdad de la operación lógica XOR.

5.5.4. El Operador Lógico NOT

Cuando se aplica el operador lógico NOT a una condición o composición de condiciones, lo que se hace es *verificar la falsedad* de la condición.

Así:

NOT *true* es lo mismo que *false* y
NOT *false* es lo mismo que *true*

El cuadro de la figura 5-10 muestra la tabla de verdad de la operación lógica NOT aplicada a una condición C1.

C1	NOT C1
SI	NO
NO	SI

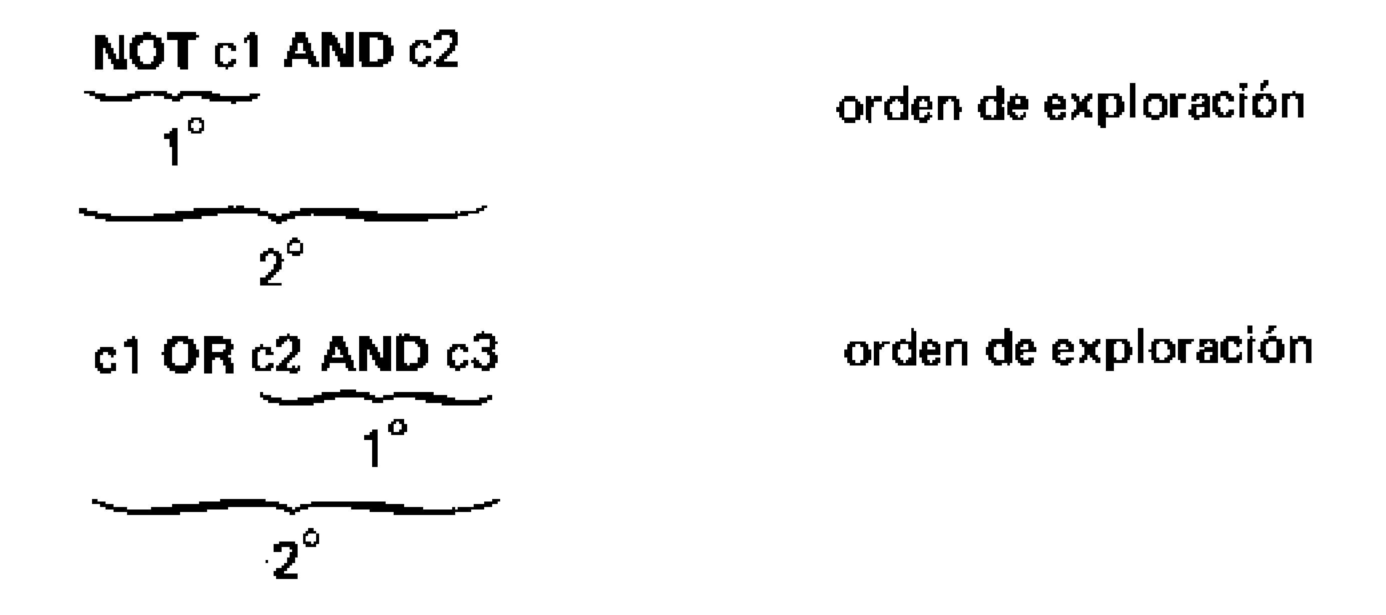
Fig. 5.10.— Tabla de verdad de la operación lógica NOT.

El operador lógico NOT puede ser aplicado sin ningún inconveniente a cualquier grupo o relación simple o compuesta.

Debe tenerse especial cuidado, no obstante, con las reglas de formación de dichas relaciones compuestas, puesto que el orden o jerarquía es:

NOT
AND
OR, XOR

Ejemplos:



Con el objeto de modificar esta jerarquía de los operadores dentro del contexto de una instrucción, pueden utilizarse los paréntesis que delimitan el orden de exploración de las condiciones.

NOT (c1 **AND** c2)
(c1 **OR** c2) **AND** c3
para los ejemplos anteriores.

No obstante, algunas combinaciones pueden ser *equivalentes*. Así, aplicando las leyes de Morgan del álgebra se tiene:

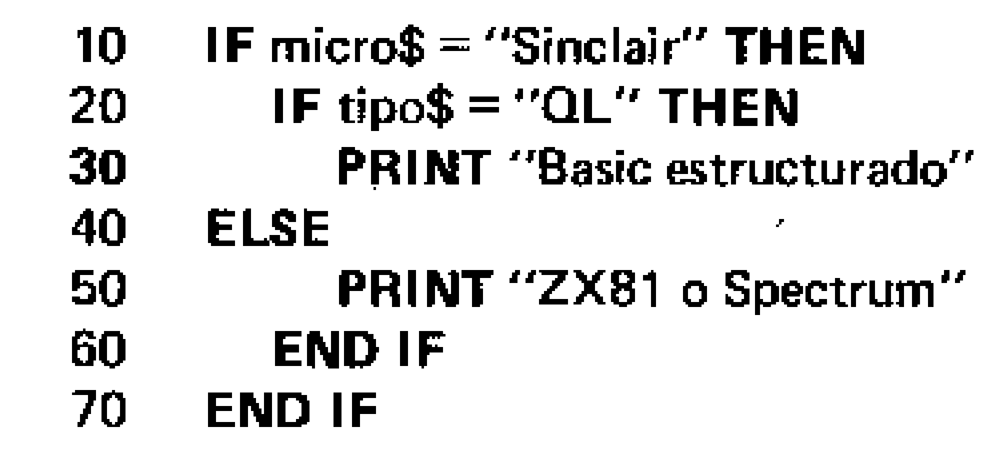
NOT (c1 **AND** c2) es equivalente a **NOT** c1 **OR NOT** c2
NOT (c1 **OR** c2) es equivalente a **NOT** c1 **AND NOT** c2

El lector interesado en profundizar puede consultar la bibliografía específica de *Algebra-moderna.*

5.6. ANIDAMIENTO DE INSTRUCCIONES IF

Tanto dentro del bloque-THEN como del bloque-ELSE de una instrucción IF pueden escribirse otras instrucciones IF hasta cualquier profundidad, respetando las limitaciones de memoria del QL.

Sea, por ejemplo, el siguiente programa:



```
10   IF micro$ = "Sinclair" THEN
20      IF tipo$ = "QL" THEN
30         PRINT "Basic estructurado"
40   ELSE
50         PRINT "ZX81 o Spectrum"
60      END IF
70   END IF
```

Este programa corresponde al organigrama de la figura 5-11.

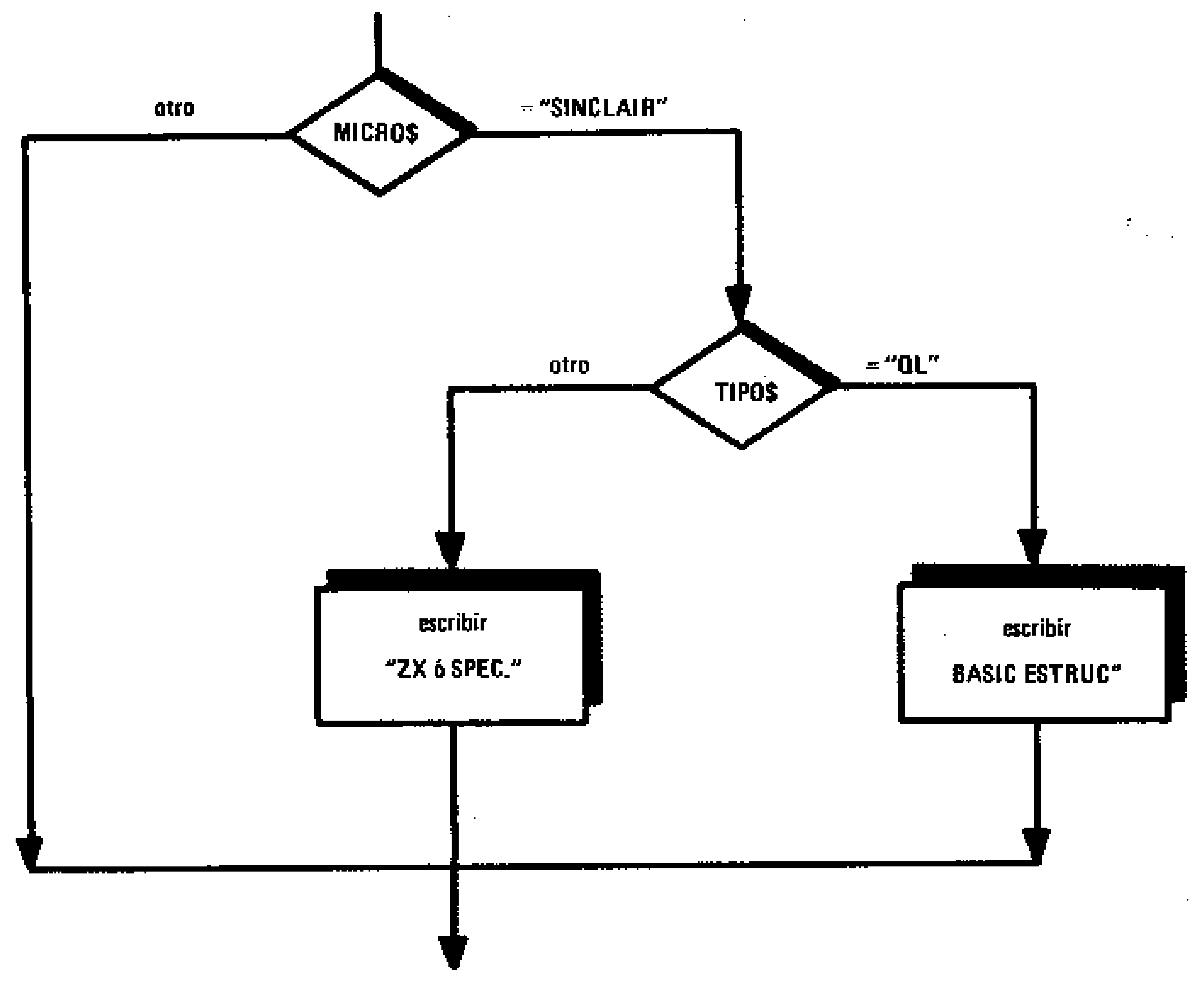


Fig. 5.11.— Ejemplo de anidamientos de instrucciones IF.

Obsérvese en el programa anterior que el bloque-THEN correspondiente a la primera instrucción IF es también otra sentencia IF con dos ramas (THEN y ELSE), es por esta razón por la que se hace necesario incluir dos END IF, el primero de ellos indicará el final del IF más interior y el segundo END IF indicará el final del IF exterior.

5.7. ELECCION ENTRE ALTERNATIVAS: La instrucción SELECT

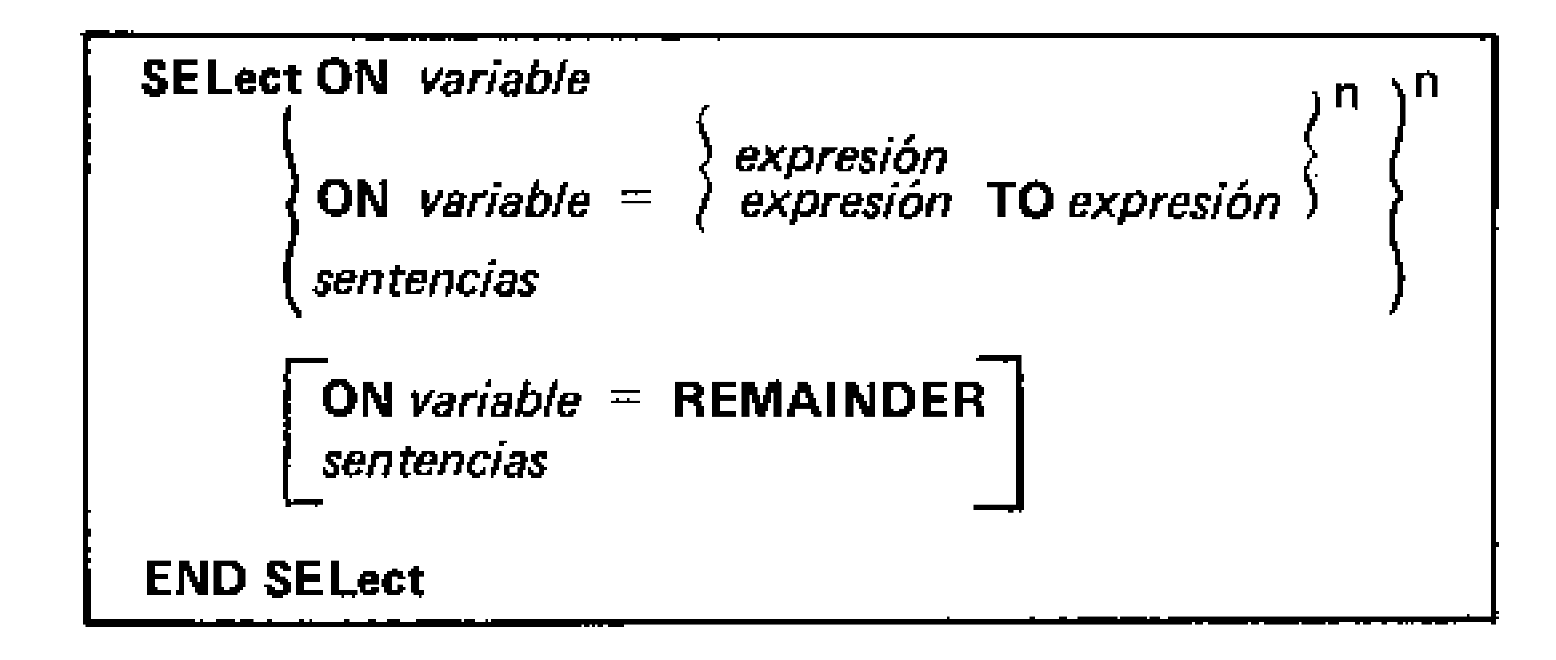
La instrucción SELect es una sentencia de programación estructurada similar en cuanto a función a la instrucción IF con la diferencia que del resultado de la evaluación de una condición pueden derivarse dos o más acciones a realizar.

La instrucción SELect comienza con esta palabra y termina con las palabras END SELect y permite, como hemos dicho, escoger de entre un posible gran número de alternativas.

Así, cuando se evalúa la condición impuesta y coincide con la primera de las alternativas se procede a ejecutar la instrucción o grupo de instrucciones correspondientes a esa alternativa. Cuando coincide con la segunda se realiza lo mismo con sus instrucciones, etc.

La instrucción SELect puede ser complementada con la instrucción ON REMAINDER asociada a un grupo de instrucciones que se ejecutarán si y sólo si no se verifica la condición para *ninguna* de las alternativas anteriores.

El formato más general de esta sentencia es:


```
SELect ON variable
     { ON variable = { expresión
                     { expresión  TO expresión }n }n
     { sentencias
     [ ON variable = REMAINDER ]
     [ sentencias              ]
END SELect
```

Sea, por ejemplo, el programa siguiente:

```
10   SELect ON   cat$
20          ON   cat$ = "director"
30               PRINT "nivel politico"
40               sueldo = 1000
50          ON   cat$ = "jefe de seccion"
60               PRINT "nivel consultivo"
70               sueldo = 500
80          ON   cat$ = "administrativo"
90               PRINT "nivel auxiliar"
100              sueldo = 100
110         ON   cat$ = REMAINDER
120              PRINT "no catalogado"
130              sueldo = 0
140  END SELect
```


Obsérvese que en este programa en la línea 10 se indica que la variable que va a ser utilizada como referenciadora de las posibles acciones a realizar es la cat$. En la línea 20 se pregunta si el contenido de dicha variable coincide con la cadena "director" y si esto es así, entonces se procede a la ejecución de las instrucciones asociadas con esa alternativa, es decir, las correspondientes a las líneas 30 y 40. Lo propio se realiza en las líneas 50 y 80 donde se pregunta por otro posible contenido de la variable-string cat$, etc.

Cuando el contenido de la variable cat$ no coincide con "director", "jefe de sección" ni con "administrativo", entonces se pasa a ejecutar la condicion ON...REMAINDER que realiza las operaciones asociadas con dicha alternativa y termina.

La figura 5-12 muestra un organigrama de funcionamiento de instrucción SELect anterior.

Obsérvese en el programa y en la figura anteriores que cuando se cumple una cualquiera de las alternativas se ejecutan las instrucciones correspondientes a ella y se deriva el control a la siguiente instrucción después de la palabra END SELect.

También puede utilizarse la instrucción SELect (sin la terminación END SELect) para proporcionar mayor claridad a una instrucción de tipo IF.

Sea, por ejemplo, el programa:

```
10   IF sueldo > 500 AND sueldo < 1000 THEN PRINT "director"
20   IF sueldo > 100 AND sueldo < 500  THEN PRINT "jefe"
30   IF sueldo >  50 AND sueldo < 100  THEN PRINT "auxiliar"
```

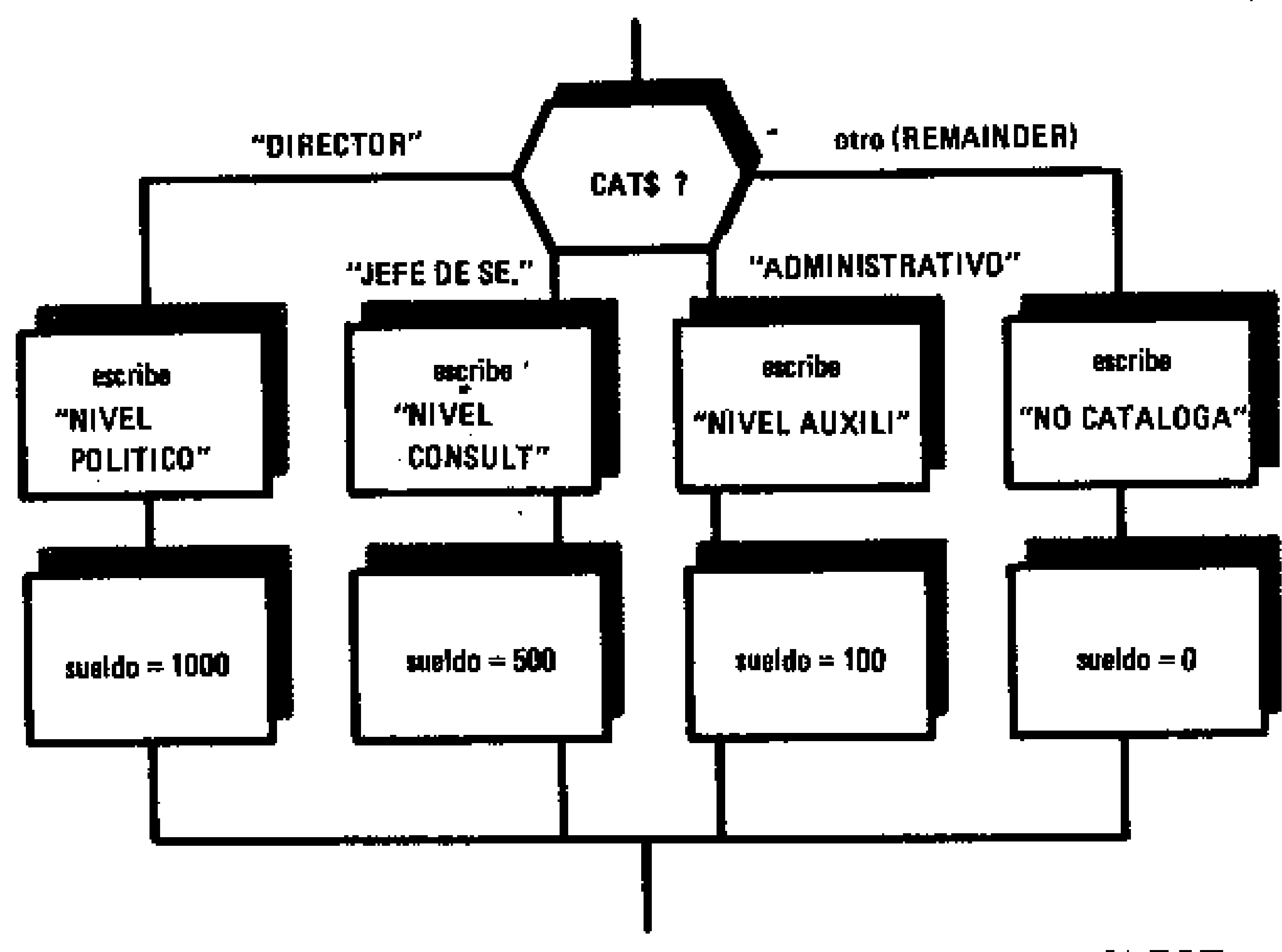


Fig. 5.12.– Ejemplo de representación de la instrucción SELECT.

Podremos utilizar la instrucción SELect para aclarar las tres sentencias anteriores de la forma:

```
10   IF SELect sueldo = 501 TO 999 THEN PRINT "director"
20   IF SELect sueldo = 101 TO 499 THEN PRINT "jefe"
30   IF SELect sueldo = 51   TO  99 THEN PRINT "auxiliar"
```

En estas tres últimas instrucciones hemos utilizado una sentencia SELect con una variable asociada (sueldo) que podrá tomar una serie de valores dentro de un determinado rango para que se ejecute el bloque-THEN de dicha instrucción IF.

Sea, por ejemplo, el programa siguiente:

```
100  numero = RND (1 TO 20)
110  SELect  ON   numero
120          ON   numero = 1
130               PRINT "es uno"
140          ON   numero = 2,4
150               PRINT "es par"
160          ON   numero = 3
170               PRINT "no hay dos sin..."
```



```
180          ON   numero = 10 TO 20
190               PRINT "segunda decena"
200          ON   numero = REMAINDER
210               PRINT "cualquiera sabe"
220  END SELect
```

Obsérvese como en la línea 140 se pregunta si la variable seleccionada *número* posee o bien el valor 2 o bien el valor 4. Por otro lado, en la línea 180 se pregunta si dicha variable posee un valor comprendido entre 10 y 20.

Con esto se ha pretendido mostrar todas las posibilidades de esta instrucción y que se desprenden de la observación del formato de la sentencia visto con anterioridad.

Ejercicio

Escribir un programa que lea cinco números y que escriba:

– "primera decena": si el número leído está comprendido entre 1 y 10.
– "segunda decena": ídem. entre 11 y 20
– El número total de múltiplos de 3 y de 5.

```
100  REMark ——————————————————
110  REMark           Multiplos
120  REMark ——————————————————
130  mult3 = 0: mult5 = 0: i = 0
140  i = i + 1
150  IF i > 5 THEN GOTO 280
160  INPUT "teclee numero" ! num
170  IF num MOD 3 = 0 THEN mult3 = mult3 + 1
180  IF num MOD 5 = 0 THEN mult5 = mult5 + 1
190  SELect  ON   num
200          ON   num = 1 TO 10
210               PRINT num ! "primera decena"
220          ON   num = 11 TO 20
230               PRINT num ! "segunda decena"
240          ON   num = REMAINDER
250               PRINT num ! "ignorado"
260  END SELect
270  GOTO 140
280  PRINT "mult. de 3=" ! mult3
```

```
290  PRINT "mult. de 5=" ! mult5
300  STOP
```

5.8. PARADA TEMPORAL: LA INSTRUCCION PAUSE

La sentencia PAUSE no presupone alteración en la ejecución secuencial de las instrucciones de un programa, sino que se efectúa una *parada temporal* de la ejecución del mismo.

El formato de la instrucción es:

PAUSE *demora*

donde *demora:* representa el tiempo que durará la pausa, expresada en unidades de 20 mseg. y puede ser o bien una constante o una variable o incluso una expresión de tipo numérico.

Si no se especifica explícitamente la duración de esta demora, el programa parará indefinidamente hasta que se produzca una entrada por el teclado.

Ejemplos:

30 PAUSE 50
espera 1 segundo

40 PAUSE 500
espera 100 segundos

60 PAUSE
espera hasta que se produzca el primer tecleo.

5.9. PARADA DEFINITIVA: LA INSTRUCCION STOP

Cuando se ejecuta una instrucción de este tipo, se termina la ejecución del programa y el sistema operativo QDOS devuelve el control al intérprete de SuperBASIC de comandos.

El formato de la sentencia es simple.

STOP

Ejemplos:

150 STOP
provocará la parada definitiva del programa en la línea en la que se encuentra.

220 IF a = 10 **THEN STOP**
Una sentencia STOP puede ser utilizada dentro de otra instrucción tal y como se muestra en la línea 220 anterior.

En cualquier caso, y en ausencia de una instrucción STOP, la última línea de un programa provocará, una vez ejecutada, una parada del mismo (salvo que se trate de una instrucción de bifurcación).

6

Instrucciones repetitivas

6.1 INTRODUCCION

Sucede con mucha frecuencia en programación práctica que se hace necesario repetir un número de veces una instrucción o grupo de instrucciones. Esta repetición se efectúa habitualmente utilizando un control en forma de condición que determina cuando ha de ejecutarse de nuevo un bucle o bien cuando se ha de salir de él.

Todo esto puede realizarse con el uso de las instrucciones estudiadas hasta ahora (por ejemplo, en el caso más sencillo, con una sentencia IF y un contador), pero hacerlo así supone no sólo una pérdida de claridad del programa sino, además, tener que utilizar indefectiblemente instrucciones del tipo GOTO con bifurcaciones dispares y con la pérdida de la potencialidad de la programación estructurada.

Con el objeto de dotar al SuperBASIC, al igual que a otros lenguajes de tipo estructurado, de estas características de programación metódica se han incluido en su definición sintáctica varias instrucciones para control de ciclos o instrucciones repetitivas que permiten realizar un grupo de instrucciones un número *controlado* de veces sin la utilización explícita de saltos incondicionales y aprovechando las características de la programación estructurada.

Estudiaremos en este capítulo dos instrucciones de este tipo, cada una de ellas con sus peculiaridades y características específicas.

La primera de ellas es la sentencia FOR que se utilizará para repetir una instrucción o grupo de instrucciones cuando se posee cierto cono-

cimiento del número de las iteraciones a efectuar. Estudiaremos las dos variantes que posee en SuperBASIC, esto es: la FOR/NEXT y FOR/END FOR.

La segunda de ellas es una instrucción eminentemente de programación estructurada y se trata de la REPEAT que se utilizará para ejecutar una o varias sentencias cuando no se posee a priori conocimiento sobre el número de iteraciones a efectuar. Estudiaremos con detalle la potencia de esta instrucción en su conjunción con la sentencia EXIT.

6.2 LA INSTRUCCION FOR

La instrucción FOR se utiliza para permitir al usuario la ejecución reiterada de determinadas instrucciones un número controlado de veces sin necesidad de escribir esas instrucciones más que una sola vez dentro del programa. Las instrucciones FOR y NEXT, o bien FOR y END FOR delimitan un *bucle* de forma que las sentencias que se encuentran entre ellas se ejecuten un número definido y controlado de veces.

La sentencia FOR consta de una variable índice (llamada *variable controlada*), de un *valor inicial* para dicha variable que se toma en el momento de entrar al bucle, un *incremento (step)* que decidirá en *cuanto* el valor de la variable controlada deberá ser incrementado (o decrementado si el step es negativo) cada vez que se ejecute la sentencia NEXT o END FOR que delimita el bucle, y un *valor final* para dicha variable controlada que determina la finalización de la ejecución de las sentencias del bucle cuando la variable controlada excede de dicho valor final.

El formato general de una instrucción FOR es el siguiente:

```
                   { exp-num                            }n
FOR variable =     { exp-num TO exp-num                 }
                   { exp-num TO exp-num STEP exp-num    }

      { instrucción }n

{ END FOR variable }
{ NEXT variable    }
```

Sea el siguiente programa ejemplo:

```
10    FOR i = 1 TO 5 STEP 1
20          PRINT i
30    NEXT i
40    PRINT "se acabo el bucle"
```

La ejecución del programa anterior dará como resultado:

```
1
2
3
4
5
se acabó el bucle
```

Cuando se llega a la ejecución de la línea 10 la variable controlada *i* toma el valor 1 sabiendo que el mayor valor que podrá tomar será 5 y en incrementos de 1 en 1. Se ejecutará, por tanto, la instrucción PRINT de la línea 20 y se llega al final del bucle con la sentencia NEXT i que incrementará dicha variable controlada según lo establecido en el STEP de la línea 10 y pasándose seguidamente a ejecutar de nuevo las instrucciones del bucle hasta que la variable controlada tome el valor 6 en cuyo caso se suspende la ejecución del bucle y se deriva el control a la siguiente instrucción después del bucle; esto es, a la línea 40 que imprime el mensaje dado.

La forma de ejecución de una instrucción FOR/NEXT o su equivalente FOR/END FOR se muestra gráficamente en la figura 6.1.

Cuando el incremento de una instrucción FOR es 1, entonces puede omitirse la parte STEP.

El programa siguiente es equivalente al anterior:

```
10    FOR i = 1 TO 5
20          PRINT i
30    NEXT i
40    PRINT "se acabo el bucle"
```

El incremento en una instrucción FOR puede ser negativo. Veámoslo con un ejemplo.

```
10    FOR a = b * c TO n + 1 STEP - 1
20          PRINT a
30    NEXT a
40    PRINT "bucle descendente"
```

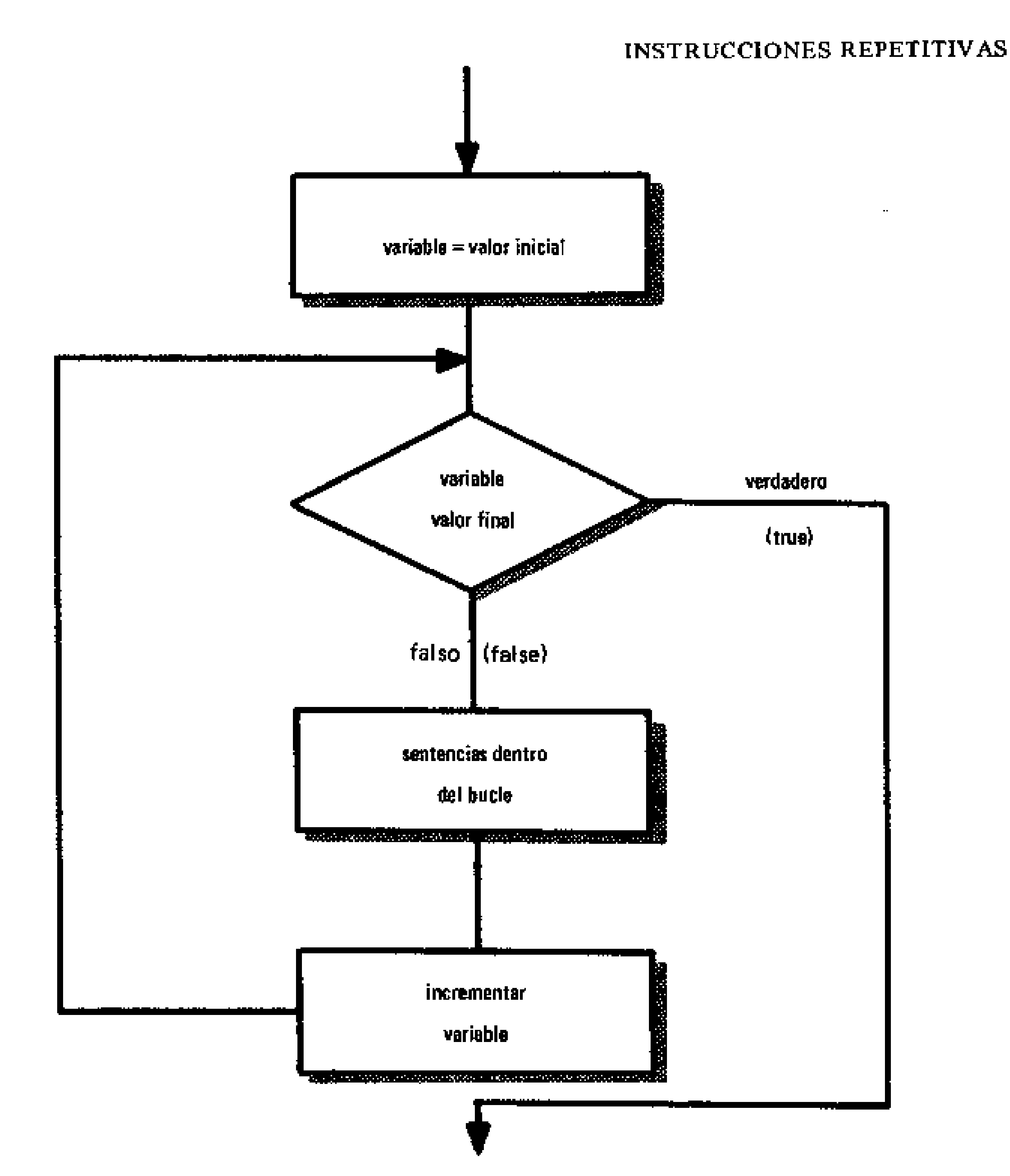



Fig. 6.1.— Ejemplo de estructura de la instrucción FOR.

Si en el programa anterior suponemos que al principio de la ejecución del bucle los valores de las variables mencionadas son b = 2, c = 3, n = 2; entonces el resultado de su ejecución será:

```
6
5
4
3
bucle descendente
```

En éste último caso la ejecución del bucle continuará hasta que el valor de A sea menor que N + 1.

Cuando *antes* de proceder a la ejecución de un bucle se tiene que *ya* se cumple la condición de terminación, entonces el bucle *no* se ejecutará ninguna vez y se derivará el control a la siguiente instrucción después de la NEXT o END FOR.

El valor de la variable controlada puede ser utilizado sin inconvenientes dentro del bucle. De igual forma esta variable controlada puede ser modificada dentro del ámbito de las instrucciones FOR, por ello si el valor de la variable controlada se cambia, el nuevo valor reemplazará al antiguo y se sumará el incremento al nuevo valor obtenido para decidir el siguiente valor de la iteración.

Las instrucciones de bifurcación condicional o incondicional pueden utilizarse dentro de un bucle de forma que siempre está permitido bifurcar hacia el *exterior* del bucle. El caso contrario, cuando se bifurca desde fuera del bucle hacia dentro del mismo ha de tenerse cuidado puesto que el valor inicial y final de la variable controlada y el incremento serán calculados solamente si se ejecuta la instrucción FOR propiamente dicha.

Ha de tenerse especial cuidado también cuando se use una instrucción GOSUB dentro de un bucle FOR dado que el retorno del control por la sentencia RETURN puede transferir el control de fuera a dentro del bucle o incluso de un bucle a otro.

Hemos visto la utilización de la sentencia tradicional NEXT como finalización de los bucles. Lo mismo puede decirse de la instrucción END FOR.

```
100  FOR i = 1 TO 5
110       PRINT i
120  END FOR i
130  PRINT "final"

     1
     2
     3
     4
     5
     final
```

Cuando dentro de un ámbito FOR sólo existe una instrucción, como en el caso anterior, puede utilizarse en SuperBASIC el formato corto de la instrucción de la manera:


```
100  FOR i = 1 TO 5: PRINT i
110  PRINT "final"
```

que provoca la misma salida que el programa anterior.

Los bucles FOR pueden estar anidados a tantos niveles como permita la situación de la memoria en un instante considerado. El *anidamiento (nested)* de instrucciones FOR significa que dentro del ámbito del bucle de una instrucción de tal tipo pueda escribirse otra instrucción de bucle.

Es muy importante hacer notar que este anidamiento debe ser estricto, esto es, que cada pareja asociada FOR/END FOR debe de estar completamente incluidos o completamente excluidos de cualquier otro bucle FOR.

Los bucles FOR, en definitiva, no pueden estar cruzados ni solapados parcialmente.

La figura 6.2 muestra un esquema de anidamientos válidos e incorrectos de instrucciones FOR/END FOR.

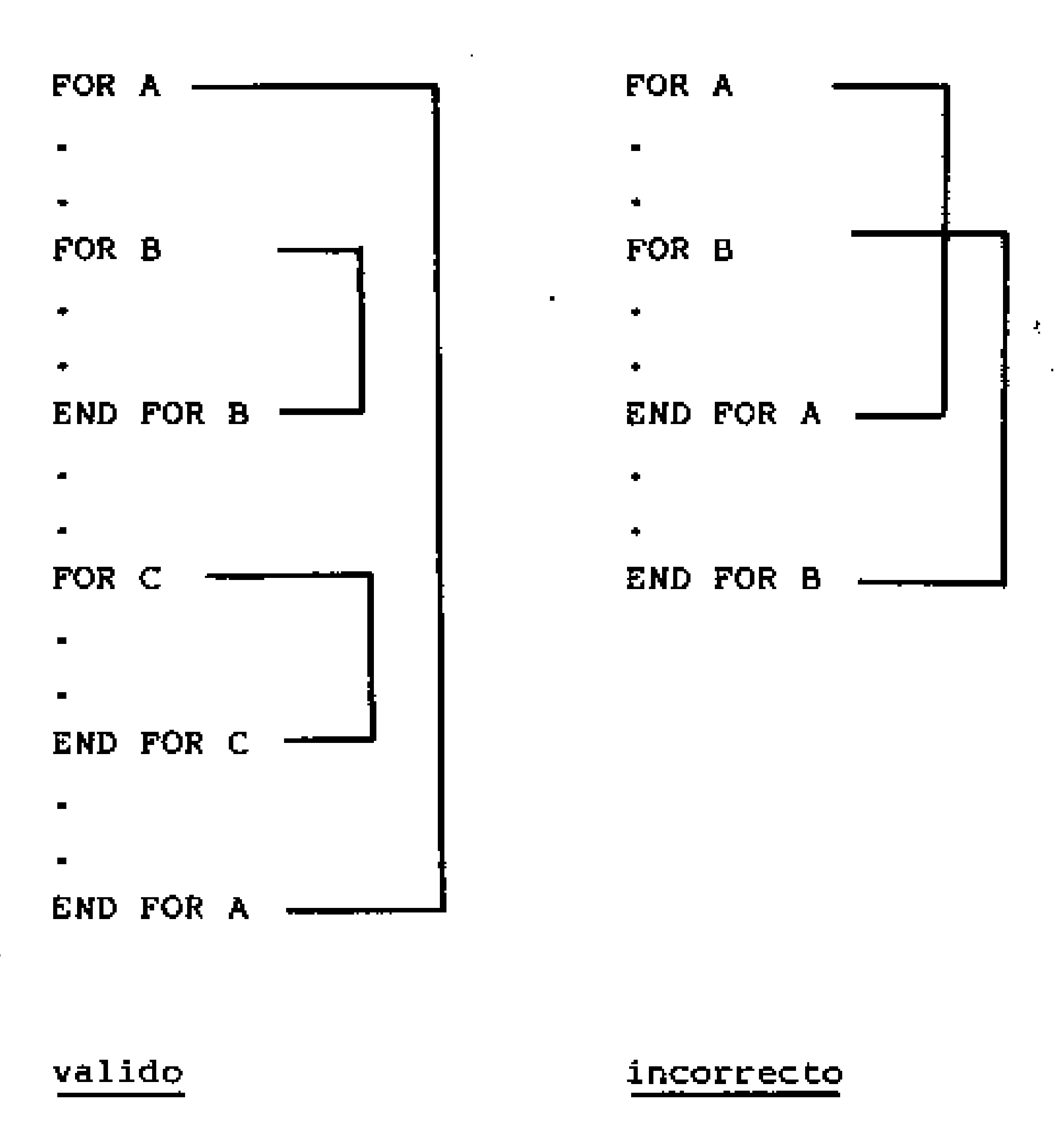


Fig. 6.2.– Ejemplos de bucles FOR correctos e incorrectos.

Dentro de la estructura de una instrucción FOR pueden mencionarse explícitamente los valores que debe tomar la variable controlada, tal y como se especificaba en el formato general.

Ejemplo

El programa siguiente:

```
100 REMark Numeros no-primos
110 FOR num = 4, 6, 8 TO 10, 12, 14 TO 16, 18, 20
120       PRINT ! num !
130 END FOR num
```

visualizará los números que no son primos de entre los veinte primeros números enteros.

Obsérvese la estructura especial que puede tomar la instrucción FOR en casos como este donde el intervalo de actuación de la variable controlada *no es uniforme.*

Dentro de un bucle FOR también puede escribirse una sentencia EXIT. Cuando dentro de una iteración se llega a la ejecución de una instrucción EXIT, entonces el control del programa bifurcará a la instrucción siguiente (NEXT o END FOR) que cierra el bucle. Veamos un

Ejemplo

```
100 suma = 0: numero = 0
110 INPUT "se debe parar cuando la suma sea? = "; numero$
120 FOR numero = 1 TO numero$
130       suma = suma + numero
140       IF suma = numero$ THEN EXIT numero
150 END FOR numero
160 PRINT "La suma es ="; suma
```

En el programa anterior, la línea 100 pone a cero las variables mencionadas. En la línea 110 se pregunta al usuario hasta qué número debe hacerse la suma. La línea 120 da comienzo al bucle FOR y es un buen ejemplo de la conversión (coerción) de la que hablamos en el capítulo 2 dado que el límite superior de la sentencia FOR es una variable de string.

La línea 130 realiza una suma depositando el contenido en la variable real SUMA. La línea 140 realiza una verificación de la llegada al límite superior de la instrucción FOR, si esto sucede, el control se deriva a la siguiente sentencia después de la END FOR, esto es; a la PRINT de la línea 160.

Por último, la línea 150 contiene el END FOR que incrementará la variable controlada y continuará el bucle de forma habitual.

El uso de la instrucción EXIT proporciona un método auxiliar para salir fuera del ámbito de una sentencia FOR como hemos visto en el programa anterior.

Ejercicio 1

Escribir un programa que lea 20 cantidades representativas de otros tantos radios de circunferencia y que calcule y escriba las longitudes de estas 20 circunferencias.

```
100 REMark ------------------
110 REMark      Circunferencias
120 REMark ------------------
130 FOR i = 1 TO 20
140       INPUT "radio?" ! radio !
150       longitud = 2 * PI * radio
160       PRINT "longitud ="; longitud
170 END FOR i
180 STOP
```

La función PI que se escribe en la línea 150 será estudiada con detalle en el capítulo 8. No obstante diremos ahora que PI representa el valor de π, esto es, aproximadamente 3,14159.

Ejercicio 2

Escribir un programa que calcule la expresión:

$$\text{suma} = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$$

dado el número n.

```
100 REMark ----------------------
110 REMark      Cálculo de una serie
120 REMark ----------------------
130 suma = 0
140 INPUT "teclee n" ! numero
150 FOR i = 1 TO numero : suma = suma + 1 / i
160 PRINT "suma =" ! suma
170 STOP
```

Ejercicio 3

Escribir un programa que calcule el número *e* base de los logaritmos naturales según la serie:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots$$

con una aproximación de 19 sumandos se considera suficiente.

```
100 REMark -------------------
110 REMark      Calculo de e
120 REMark -------------------
130 e = 1
140 t = 1
150 FOR n = 1 TO 19
160    t = t / n
170    e = e + t
180 END FOR n
190 PRINT "e =" ! e
200 STOP
```

Ejercicio 4

Escribir un programa que lea 20 parejas de cantidades representativas de las edades de cada uno de los 20 empleados de una empresa.

Se pretende sumar los sueldos de aquellos empleados cuyas edades estén comprendidas entre 20 y 35 años.

```
100 REMark -------------------
110 REMark      Suma de sueldos
120 REMark -------------------
130 suma = 0
140 FOR i = 1 TO 20
150       INPUT "datos?" ! edad, sueldo
160       IF edad < 20 OR edad > 35 THEN
170          PRINT "edad no correcta"
180       ELSE
190          suma = suma + sueldo
200       END IF
210 END FOR i
220 PRINT "la suma es =" ! suma
230 STOP
```


Ejercicio 5

Escribir un programa que llene la matriz de la figura 4.4 sabiendo que se lee un número cada vez y por filas.

```
100 REMark -------------------
110 REMark      Llenado de una matriz
120 REMark -------------------
130 DIMensión vehículos(3, 4)
140 FOR fila = 1 TO 3
150       FOR columna = 1 TO 4
160          INPUT "dato?" ! dato
170          vehículos(fila, columna) = dato
180       END FOR columna
190 END FOR fila
200 STOP
```

Obsérvese que en el programa anterior no se ha utilizado los subíndices cero de cada una de las dimensiones de la matriz.

6.3 LA INSTRUCCION REPEAT

La sentencia REPeat se utiliza para repetir un número controlado de veces una o más instrucciones.

El formato de la instrucción es:

```
REPeat identificador
        {instrucciones}^n
END REPeat identificador
```

donde *identificador* debe estar formado según las reglas de formación ya vistas en el capítulo 2.

Las instrucciones contenidas dentro de una sentencia REPeat se ejecutarán cíclicamente hasta que se cancele su ejecución apretando simultáneamente las teclas CTRL y la barra de espaciado (break) o bien hasta que se le de una salida lógica al bucle mediante una instrucción EXIT.

Sea el siguiente programa:

```
10   PAPER 7
20   BORDER 3, 2
30   REPeat bucle
40        INK RND(5)
50        LINE 120, 60 TO RND(100), RND(100)
60   END REPeat bucle
```

El programa anterior dibuja una estrella con origen en el punto 120, 60 y donde cada una de las líneas posee un color diferente.

Obsérvese el uso de la instrucción REPeat y cómo necesariamente ha de abortarse la ejecución de este programa con el objeto de detenerlo, (CTRL + espacio).

Una variante del mismo se muestra a continuación:

```
10   PAPER 7
20   BORDER 3, 2
30   i = 0
40   REPeat bucle
50        INK RND(5)
60        LINE 120, 60 TO RND(100), RND(100)
70        i = 1 + 1
80        IF i > 50 THEN EXIT bucle
90   END REPeat bucle
```

Obsérvese cómo en esta variante se añade una instrucción condicional IF que controla el número de veces que se realiza el bucle. Cuando la condición se cumple, se sale fuera del bucle (EXIT) y se pasaría a ejecutar la siguiente instrucción después de la END REPeat. En este caso y como no existen más sentencias a ejecutar, el programa parará.

Las instrucciones FOR y REPeat pueden estar anidadas. El programa siguiente ilustra esta afirmación y obtiene puntuaciones de dos dados, una tira por fila, hasta que se consigue el valor 8. Se toman cuatro filas.

```
100  REMark Puntuaciones con dados
110  FOR fila = 1 TO 4
120       PRINT "fila numero" ! fila
130       REPeat tirada
140            dado1 = RND(1 TO 6)
150            dado2 = RND(1 TO 6)
160            puntos = dado1 + dado2
```



```
170            PRINT puntos !
180            IF puntos = 8 THEN EXIT tirada
190       END REPeat tirada
200       PRINT \"final de la fila" ! fila
210  END FOR fila
```

Una posible salida de este programa sería:

```
fila numero 1
7699710610636676101 1768
final de la fila 1
fila numero 2
26628
final de la fila 2
fila numero 3
628
final de la fila 3
fila numero 4
61091095574977444310 74648
final de la fila 4
```

Obsérvese como ha sido utilizada la instrucción EXIT para salir del ámbito del bucle. La situación más frecuente es la que se muestra en el programa anterior, donde esta salida está condicionada y controlada por una instrucción IF.

Ejercicio 1

Escribir un programa que lea un número entero dentro del rango 1 a 3.999 y que lo transforme a su equivalente en números romanos.

```
100  REMark -------------------
110  REMark      Numeros romanos
120  REMark -------------------
130  INPUT "introducir numero" ! num
140  FOR tipo = 1 TO 7
150       READ letra$, valor
160       REPeat salida
170            IF num < valor THEN EXIT salida
```

```
180             PRINT letra$
190             num = num – valor
200          END REPeat salida
210   END FOR tipo
220   DATA "M", 1000, "D", 500, "C", 100, "L", 50
230   DATA "X", 10, "V", 5, "I", 1
```

Ejercicio 2

Escribir un programa que calcule x^y debiendo ser los números x e y mayores que cero.

```
100  REMark ------------------
110  REMark      Calculo de potencias
120  REMark ------------------
130  INPUT "teclee los x e y" ! x ! y
140  IF x <= 0 OR y <= 0 THEN GOTO 130
150  contador = 0
160  resultado = 1
170  REPeat bucle
180        IF contador > = y THEN EXIT bucle
190        resultado = resultado * x
200        contador = contador + 1
210  END REPeat bucle
220  PRINT "x elevado a y =" ! resultado
230  STOP
```

Ejercicio 3

Calcular el factorial de un número leído previamente, utilizando la sentencia REPeat.

```
100  REMark ------------------
110  REMark      Factorial
120  REMark ------------------
130  INPUT "teclear numero" ! numero
140  fact = 1
150  REPeat calculo
160        IF numero < = 0 THEN EXIT calculo
170        fact = fact * numero
180        numero = numero – 1
190  END REPeat calculo
200  PRINT "el resultado es =" ! fact
210  STOP
```


Ejercicio 4

Escribir un programa que lea un número indeterminado de cantidades y que calcule e imprima la suma de ellas y su media aritmética. El proceso parará cuando se lea un cero.

```
100  REMark ------------------
110  REMark      Suma y media aritmética
120  REMark ------------------
130  suma = 0 : total = 0
140  REPeat calculo
150        INPUT "numero?" ! numero
160        IF numero = 0 THEN EXIT calculo
170        total = total + 1
180        suma = suma + numero
190  END REPeat calculo
200  media = suma / total
210  PRINT "la suma es =" ! suma
220  PRINT "la media es =" ! media
230  STOP
```

Ejercicio 5

Escribir un programa que lea un número indeterminado de cantidades y que cuente el número de las que son pares y el número de las que son impares hasta que cualquiera de estos acumulados sea superior a 20.

```
100  REMark ------------------
110  REMark      Pares e impares
120  REMark ------------------
130  pares = 0 : impares = 0
140  REPeat leer
150        INPUT "numero?" ! numero
160        IF número MOD 2 = 0 THEN
170           impares = impares + 1
180        ELSE
190           pares = pares + 1
200        END IF
210        IF pares = 20 OR impares = 20 THEN EXIT leer
220  END REPeat leer
230  PRINT "pares =" ! pares
240  PRINT "impares = " ! impares
250  STOP
```

Ejercicio 6

Escribir un programa SuperBASIC que lea una cierta cantidad de números enteros, los cuente y que cuente también los números múltiplos de 3 por un lado y de 7 por otro.

```
100 REMark ------------------
110 REMark      Multiplos
120 REMark ------------------
130 total = 0:  mult3 = 0:  mult7 = 0
140 REPeat lectura
150      INPUT "numero?" ! num
160      IF num = 99 THEN EXIT lectura
170      total = total + 1
180      IF num MOD 3 = 0 THEN mult3 = mult3 + 1
190      IF num MOD 7 = 0 THEN mult7 = mult7 + 1
200 END REPeat lectura
210 PRINT "nros. leidos =" ! total
220 PRINT "múltiplos de 3 =" ! mult3
230 PRINT "múltiplos de 7 =" ! mult7
240 STOP
```

Obsérvese que el programa terminará cuando lea el número 99.

Ejercicio 7

Escribir un programa que lea pares de coordenadas (x, y) de puntos del plano. Se deben contar los puntos leidos y de ellos los que están en el interior del círculo $x^2 + y^2 = 25$.

```
100 REMark ----------------------
110 REMark      Puntos del circulo
120 REMark ----------------------
130 cont = 0:  total = 0
140 REPeat lectura
150      INPUT "x, y ?" ! x ! y
160      IF x = 99 AND y = 99 THEN EXIT lectura
170      total = total + 1
180      IF x * y + y * y <= 25 THEN cont = cont + 1
190 END REPeat lectura
200 PRINT "puntos leidos =" ! total
210 PRINT "puntos interiores =" ! cont
220 STOP
```

Este programa parará cuando se lea un punto de coordenadas (99, 99) tal y como se muestra en la línea 160.

7
Subprogramas clásicos

7.1. INTRODUCCION

Con el fin de hacer compatible el SuperBASIC del QL con otros BASIC's anteriores, el lenguaje permite el uso de instrucciones características de subprogramación como son la GOSUB para bifurcación incondicional a subrutinas y la ON...GOSUB para bifurcación condicional a subrutinas previo cumplimiento de una condición.

Cuando se estudien en el capítulo 8 las funciones *(functions)* y los procedimientos *(procedures)* se verá con claridad como el uso de las instrucciones GOSUB y ON...GOSUB no son necesarias (ni incluso convenientes) a la hora de confeccionar programas, pues son suplidas con enorme ventaja por estas modernas técnicas de programación estructurada. No obstante se ha incluido también aquí su estudio para proporcionar al lector una visión completa del lenguaje SuperBASIC.

La justificación lógica del uso de las subrutinas se plantea cuando se requiere la ejecución de cierto grupo de instrucciones en varios puntos de un mismo programa.

La solución más burda sería, evidentemente, colocar aquel grupo de instrucciones *en todos y cada uno* de los lugares en los que se requiera su ejecución. Esta solución plantea, como es lógico, multitud de inconvenientes y nunca es recomendable.

Otra solución será escribir *una sola vez* el grupo de instrucciones que componen la subrutina de forma que sean ejecutadas mediante una sola instrucción de llamada a dicha subrutina en aquellos lugares en los que se precise su ejecución.

Siguiendo esta última filosofía se pasan a describir seguidamente las instrucciones SuperBASIC para manejo de estos subprogramas.

7.2. LLAMADA INCONDICIONAL: La instrucción GOSUB

La instrucción GOSUB tiene el formato:

GOSUB *línea*

y se utiliza para ejecutar una sección de un programa definida implícitamente como una subrutina. La sentencia GOSUB transfiere el control a la *línea* referenciada en la propia instrucción y almacena la dirección de llamada de forma que cuando se ejecute la última instrucción a la subrutina se transferirá el control a la línea siguiente a la propia sentencia GOSUB.

Este retorno del control se realiza cuando se alcanza una instrucción:

RETURN

en la subrutina. Cuando se encuentra esta instrucción se transfiere el control a la línea siguiente a la GOSUB de llamada.

Sea el siguiente programa ejemplo:

```
10   a=2
20   b=0
30   GOSUB 60
40   PRINT b
50   STOP
60   b=a+b
70   IF b>6 THEN GOTO 90
80   GOSUB 60
90   RETURN
```

El primer GOSUB transfiere el control a la línea 60. Dado que B solamente es igual a 2, la sentencia GOSUB de la línea 80 se mantendrá llamando a la rutina y sumando A a B hasta que B = 8. Entonces el control pasará a ejecutar la sentencia RETURN de la línea 80 que será

la que devolverá el control al último de los GOSUB utilizados que es precisamente el que se encuentra en la línea 80. La segunda vez que se llega a la instrucción RETURN, esta devolverá el control a la segunda de la última GOSUB encontrada y por este motivo se retrocede el control hasta la línea después de la GOSUB, esto es, a la línea 40.

Sea el siguiente programa:

```
10   PRINT "Llamada a subrutina"
20   GOSUB 50
30   PRINT "Final de la ejecucion"
40   STOP
50   PRINT "Esta es la subrutina"
60   a=b +c
70   PRINT "Regreso a la llamada"
80   RETURN
```

Cuando se ejecute este programa en el QL, dará como resultado:

```
Llamada a la subrutina
Esta es la subrutina
Regreso a la llamada
Final de la ejecucion
```

El SuperBASIC del QL permite que el número de la línea de destino de una instrucción GOTO o GOSUB pueda ser el resultado de alguna expresión aritmética.

Ejemplos:

```
10   x=a/b
20   GOTO x
o bien directamente como:
30   GOSUB 8 * x
```

Desde luego, hay que tener cuidado con la utilización de este procedimiento de transferencia, pues al renumerar el programa, utilizando el comando RENUM es muy posible que se derive el control hacia un lugar equivocado, dado que este comando no afecta a las líneas definidas implícitamente como las anteriores.

El comando RENUM para la renumeración automática de las líneas de un programa SuperBASIC se estudia con detalle en el Apéndice-A dedicado a los comandos del SuperBASIC.

7.3. LA LLAMADA CONDICIONADA: La instrucción ON...GOSUB

Como ya hemos visto con anterioridad, en general, la instrucción ON se usa para transferir el control del programa a una o varias sentencias del mismo programa y que en este caso son sentencias de comienzo de subrutinas, en función del valor de una expresión.

El formato general de una instrucción ON...GOSUB es:



ON *variable* **GOSUB** *expresión* $\left[\{, expresión\}^n \right]$

donde *variable:* debe ser un identificador, y *expresión* puede ser o bien una constante o variable numéricas e incluso una expresión de tal tipo y son representativos de los números de líneas hacia las que bifurca el programa.

En el caso más simple sería:

ON *variable* **GOSUB** $\ell_1, \ell_2, ..., \ell_n$

donde l_i son los números de líneas donde dan comienzo las subrutinas.

Como siempre, si el valor de la variable es 1, el control se transfiere a la primera de las líneas especificadas en la instrucción, si la variable vale 2, el control se transfiere a la segunda línea mencionada, etc.

Ejemplo:

Sea la instrucción:

```
10   ON a GOSUB 100, 150, 200
```

Si *a* toma el valor 2, el control se transferirá a la segunda sentencia, esto es; a la línea 150, etc.

Si el valor de la variable es menor que 1 ó mayor que el número de líneas dadas, se producirá un error de ejecución.

Sea el siguiente programa:

```
10   e = RND (1 TO 3)
20   ON e GOSUB 40, 60, 80
30   GOTO 10
```



```
40   PRINT "nro. aleatorio = 1"
50   RETURN
60   PRINT "nro. aleatorio = 2"
70   RETURN
80   PRINT "nro. aleatorio = 3"
90   RETURN
```

En la línea 10 se ha utilizado la función RND que genera un número aleatorio, en este caso comprendido entre 1 y 3. Así pues, en función de la variable *e* se deriva el control a la subrutina de la línea 40 si el valor de e = 1, o bien a la subrutina de la línea 60 si el valor de e = 2, etc.

Como se deduce de todo lo anterior, tanto la instrucción GOSUB como la ON...GOSUB dependen para su sintaxis de los números de las líneas a donde se pretende bifurcar. Esto hace que los programas escritos de esta manera sean poco susceptibles de modificarse sin tener que cambiar pocas cosas.

Como veremos en el capítulo siguiente, con el uso de las funciones y los procedimientos, esta dependencia respecto de los números de línea es absolutamente nula, con lo que conseguiremos una programación totalmente independiente y estructurada.

8

Subprogramas avanzados: Funciones y procedimientos

8.1. INTRODUCCION

Como ya mencionábamos en el capítulo anterior, sucede habitualmente el hecho de que algún grupo de instrucciones tenga que repetirse en diferentes lugares del programa, con lo que sin duda se alarga el tiempo de programación y los programas se hacen también más voluminosos y aburridos de leer y de mantener.

Parece lógico, pues, suponer la existencia de algo que nos permita escribir estas instrucciones que se repiten una sola vez y que puedan ser ejecutadas siempre que el programador lo desee sin más que invocarlas. Pues bien, con algunas diferencias que estudiaremos en el presente capítulo, este algo de lo que hablamos es lo que se denomina en SuperBASIC: *funciones y procedimientos.*

Así, en general, podemos afirmar que los procedimientos y las funciones son *subprogramas* de tipo *avanzado* que se localizan en memoria una sola vez, en un lugar determinado y que pueden ser *invocados* o *llamados* todas las veces que el programador estime necesario.

La utilización de funciones y procedimientos en SuperBASIC libran al programador de la servidumbre de depender, como veremos, de los números de las líneas de programa, además de proporcionar una enorme ventaja: la transmisión de valores entre el programa principal y los subprogramas, lo que llamaremos *argumentos formales* y *argumentos reales* y que constituye la verdadera potencia del uso de las funciones y los procedimientos.

Existen también en SuperBASIC un buen número de *funciones incorporadas* de gran utilidad que no es necesario que defina el propio

usuario, ya que se encontrarán permanentemente contenidas en el repertorio del SuperBASIC.

Las hemos dividido en *funciones aleatorias* para la generación de números aleatorios, *funciones matemáticas* para su uso conjunto con constantes, variables o expresiones numéricas, funciones de logaritmos, funciones trigonométricas que proporcionan los valores angulares más usuales, *funciones de memoria* para el acceso y modificación de segmentos de memoria y las *funciones de teclado* para su uso combinado con el teclado del QL.

En los apartados siguientes se estudian con detalle las funciones y procedimientos de usuario propiamente dichas.

8.2. FUNCIONES ALEATORIAS

Estas funciones incorporadas permiten la obtención y uso de *números aleatorios.*

Los números aleatorios son cantidades obtenidas aleatoriamente, es decir, arbitrariamente con el objeto de disponer siempre de valores que son siempre desconocidos por el programador.

Algunas aplicaciones muy frecuentes de los números aleatorios son en la construcción de programas para juegos, estadística, investigación operativa, etc.

8.2.1. La función RANDOMISE

Esta función activa el generador de números aleatorios.

El formato de la función es:

RANDOMISE	[*expresión-numérica*]

Cuando no se utiliza la *expresión-numérica*, el generador de números aleatorios utilizará sus propias funciones internas para reactivar la generación.

Cuando se utiliza la *expresión-numérica*, este generador utilizará el valor de esta expresión para arrancar el proceso.

Ejemplos:

10 **RANDOMISE**
generación de números estándar

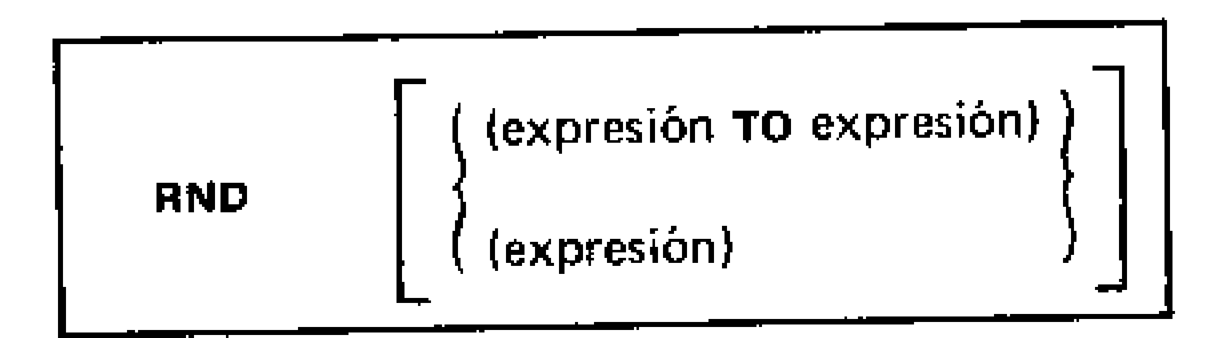
40 **RANDOMISE** 2.325
generación a partir de dicha cantidad

Si suponemos la extracción de bolas en un bombo como un ejemplo homogéneo respecto a la obtención de números aleatorios, podríamos decir que la función RANDOMISE es equivalente al movimiento de los bombos previo a la extracción de la bola. Esta extracción es propiamente equivalente a la función RND que veremos seguidamente.

8.2.2. La función RND

Esta función devuelve un *número-aleatorio* comprendido en el intervalo que se menciona.

El formato de la función es:



Ejemplos

RND (1 **TO** 25)
devuelve un número aleatorio (entero) comprendido entre 1 y 25.
Cuando el valor inicial del intervalo es cero, puede omitirse.

RND (100)
devuelve un número entero aleatorio comprendido entre 0 y 100.

Cuando la función RND se utiliza sin argumentos, entonces se devuelve un valor real comprendido entre 0 y 1.

10 **PRINT RND**
imprimirá un nro. real comprendido entre 0 y 1.

8.3. FUNCIONES MATEMATICAS

Estas funciones incorporadas trabajan de forma aritmética, proporcionando los valores más utilizados para matemáticas, como son: el *valor absoluto* de una variable o expresión, la *parte entera*, la *raíz cuadrada* y la *exponenciación*.

Veámoslas por separado.

8.3.1. La función ABS

Esta función devuelve el *valor absoluto* del resultado de la evaluación de la expresión.

El formato de la función es:

ABS *(expresión-numérica)*

Ejemplos:

ABS(24)
devuelve el valor 24

ABS(-24)
devuelve también el valor 24

```
10   a = 7
20   b = 8: c = −10
30   PRINT ABS(b -a * c)
```
imprimirá el valor 62

En síntesis, lo que hace esta función, es encontrar el valor *positivo* de la expresión, constante o variable mencionada como argumento.

8.3.2. La función INT

Esta función devuelve la parte entera del resultado de la evaluación de la expresión mencionada.

El formato de la función es:

INT *(expresión-numérica)*

Ejemplos:

INT (3.4)
devuelve el valor 3

INT (8.6)
devuelve el valor 8

INT (0.6)
devuelve el valor 0

Esta función puede ser utilizada para obtener el resultado *redondeado* del entero más próximo de la evaluación de una variable o expresión en general.

```
10   REMark   Redondeo
20   INPUT    "teclear un numero" ! num
30   PRINT    "redondeado" ! INT (num + 0.5)
```

8.3.3. La función SQRT

Esta función devuelve la *raíz cuadrada* del resultado de la evaluación de la expresión mencionada.

El formato de la función es:

SQRT *(expresión-numérica)*

donde, como siempre, *expresión-numérica* puede ser o bien una constante o una variable numéricas o incluso una expresión del mismo tipo.

Ejemplos:

SQRT (2)
devuelve el valor 1.414214

SQRT (25)
devuelve el valor 5

SQRT (2.6)
devuelve el valor 1.612452

10 **PRINT SQRT** (–1)
provocara un error de ejecucion

De lo anterior se desprende que el argumento de esta función debe ser *mayor o igual que cero* para que se obtengan resultados satisfactorios.

Puede simularse el uso de la función SQRT escribiendo un programa que realice este cometido por aproximaciones sucesivas.

```
100  REMark  Raices Cuadradas
110  INPUT  "teclee el numero" ! num
120  aprox = num /2
130  REPeat  raíz
140        nuevo = (aprox + num/aprox) / 2
150        IF nuevo == aprox THEN EXIT raiz
160        aprox = nuevo
170  END REPeat
180  PRINT "la raiz cuadrada es =" ! nuevo
```

El signo == de la línea 150 puede interpretarse como "aproximadamente igual" y dará como resultado *true* dentro del intervalo 0.0000001.

8.3.4. La función EXP

Esta función devuelve la *exponenciación* e^x

El formato de la función es:

EXP *(expresión-numérica)*

donde *expresión-numérica* debe poseer un valor comprendido dentro del intervalo –500 a 500.

Ejemplos:

EXP (2.718182)
devolvera el valor 1

EXP (256)
devolverá el valor 1.511428E111

EXP (25 + 32 * 6.5)
devolvera el valor 1.551009E101

Como el lector ya sabe de matemáticas básicas, el número *e* es la base de los logaritmos naturales o neperianos y su valor es 2.718182 aproximadamente.

8.3.5. Funciones de logaritmos: LN y LOG10

El SuperBASIC del QL dispone de dos funciones para el cálculo de los *logaritmos* habituales utilizados en matemáticas.

LN *(expresión-numérica)*

Que devuelve el *logaritmo natural* (en base e) de la expresión encerrada entre paréntesis, y

LOG10 *(expresión numérica)*

Que devuelve el *logaritmo común* (en base 10) de la expresión encerrada entre paréntesis.

En ambos casos *expresión-numérica* debe ser un valor mayor que cero.

Ejemplos:

10 **PRINT LN** (4.32)
imprimira el logaritmo natural de 4.32 = 1.463255

20 **PRINT LOG10 (10 + 3)**
imprimira el logaritmo comun de 13. = 1.113943

8.4. FUNCIONES TRIGONOMETRICAS

El SuperBASIC del QL dispone de funciones trigonométricas incorporadas que contemplan los casos más usuales de manejo de expresiones angulares.

La tabla de la figura 8.1 muestra el formato de estas funciones y el rango de aplicación de cada una de ellas.

FORMATO	FUNCION	RANGO EVALUABLE
SIN(expresion)	seno	entre -60000 y 60000
COS(expresion)	coseno	entre -60000 y 60000
TAN(expresion)	tangente	entre -30000 y 30000
COT(expresion)	cotangente	entre -30000 y 30000
ASIN(expresion)	arcoseno	sin limites explicitos
ACOS(expresion)	arcocoseno	"
ATAN(expresion)	arcotangente	"
ACOT(expresion)	arcocotangente	"

Fig. 8.1.— Las funciones trigonométricas.

La *expresión-numérica* que puede utilizarse en cada uno de los formatos representa el *ángulo* al que debe ser aplicada la función correspondiente. Puede ser o bien una constante, o una variable o incluso, como decíamos, una expresión, pero en cualquier caso este ángulo deberá ser expresado en *radianes.*

El SuperBASIC posee también la función incorporada PI. Esta función no posee argumento alguno y devuelve el valor de π cuando es utilizada.

El formato de la función es simple:

PI

Ejemplo:

20 **PRINT PI**
imprimira 3.14159...

Como siempre, una función puede usarse como argumento de otra función más exterior. Este es el caso de la función PI cuando se trabaja con funciones trigonométricas de argumentos expresados en radianes.

Ejemplos:

20 PRINT SIN (PI/2)
imprimira el seno de $\pi/2$ radianes, esto es; el seno de 90 grados

Observe el lector que todas las funciones trigonométricas anteriores operan con parámetros angulares expresados en radianes. El SuperBASIC del QL dispone de una función incorporada para conversión de dichos valores.

El formato de la función es:

DEG *(expresión-numérica)*

donde *expresión numérica* debe ser una cantidad que represente radianes.

El resultado de la ejecución de una función DEG es la conversión de radianes a *grados.*

Ejemplo:

30 PRINT DEG (PI/2)
dado que **PI** es una constante en SuperBASIC, el resultado de la instrucción PRINT anterior daría como resultado 90 de forma que:

$\pi/2$ radianes = 90 grados

El SuperBASIC del QL dispone también de la función inversa a la DEG estudiada con anterioridad.

La función incorporada RAD *convierte* un ángulo especificado en grados a un ángulo especificado en *radianes.*

El formato de la función es:

RAD *(expresión-numérica)*

donde *expresión-numérica* debe ser una cantidad representativa del ángulo que debe estar especificada en grados.

Ejemplos:

30 PRINT RAD (180)
imprimira el valor 3.141593 ya que:
180 grados = π radianes

8.5. LAS FUNCIONES DE MEMORIA

El SuperBASIC del QL dispone de varias funciones incorporadas que manejan posiciones de memoria con el objeto de acceder a ellas y modificarlas, si esto es preciso.

8.5.1. La función PEEK

La función PEEK devuelve el *contenido* de una localización específica de memoria.

Esta función PEEK posee tres formatos de actuación según la cantidad de memoria accedida.

PEEK	*(dirección):*	accede *a un byte* (8 bits)
PEEK_W	*(dirección):*	accede *a una palabra* (16 bits)
PEEK_L	*(dirección):*	accede *a una palabra larga* (32 bits)

Tanto para utilizar PEEK_W y PEEK_L, la dirección especificada en el argumento debe ser una dirección *par.*

Ejemplos:

30 PRINT PEEK (10435)
imprime el byte que posee la dirección 10435

40 PRINT PEEK_W (14)
imprime la palabra de direcciones 14 y 15

60 PRINT PEEK_L (1000)
imprime la palabra-larga de direcciones 1000, 1001, 1002 y 1003.

8.5.2. La función POKE

La función POKE permite *modificar* zonas de memoria.

Esta función POKE posee tres formatos de actuación según la cantidad de memoria susceptible de ser modificada.

POKE *dirección, dato*	: acceso a un *byte* (8 bits)
POKE_W *dirección, dato*	: acceso a una *palabra* (16 bits)
POKE_L *dirección, dato*	: acceso a una *palabra-larga* (32 bits)

Tanto para utilizar POKE_W como POKE_L, la *dirección* especificada en el argumento debe ser una dirección *par*.

Dato es la cantidad representativa de lo que hay que insertar en la *dirección* especificada y puede ser bien una constante, una variable o incluso una expresión.

Ejemplos:

100 **POKE** 10435,0
llena a ceros el byte de dirección 10435

120 **POKE_L** 128060,98765
llena con la cantidad 98765 la palabra-larga de direccion 128060

8.6. FUNCIONES DE TECLADO

El SuperBASIC dispone de dos funciones de teclado que pueden activarse desde el propio teclado de la máquina. Se trata de las funciones INKEY$ y la KEYROW. Véamoslas por separado.

8.6.1. La función INKEY$

Esta es una función no numérica que devuelve un símbolo o carácter de entrada por teclado o por el canal especificado en la propia instrucción.

El formato de la función es:

INKEY$ [*(canal* [, *tiempo*]*)*]

donde *tiempo:* es el intervalo de tiempo que esperará la instrucción en la que está imbuida la función INKEY$ antes de retornar al programa. Si no se especifica este parámetro, entonces el control se devolverá inmediatamente.
La codificación empleada en este parámetro es:
de 1 a 32768: espera el número de unidades de tiempo especificadas.
–1: espera por tiempo indefinido.
0: regresa inmediatamente.

Ejemplos:

10 **PRINT INKEY$**
imprimira el caracter apretado desde el canal por defecto (teclado)

30 **PRINT INKEY$** (40)
esperara 40 unidades de tiempo antes de devolver el control

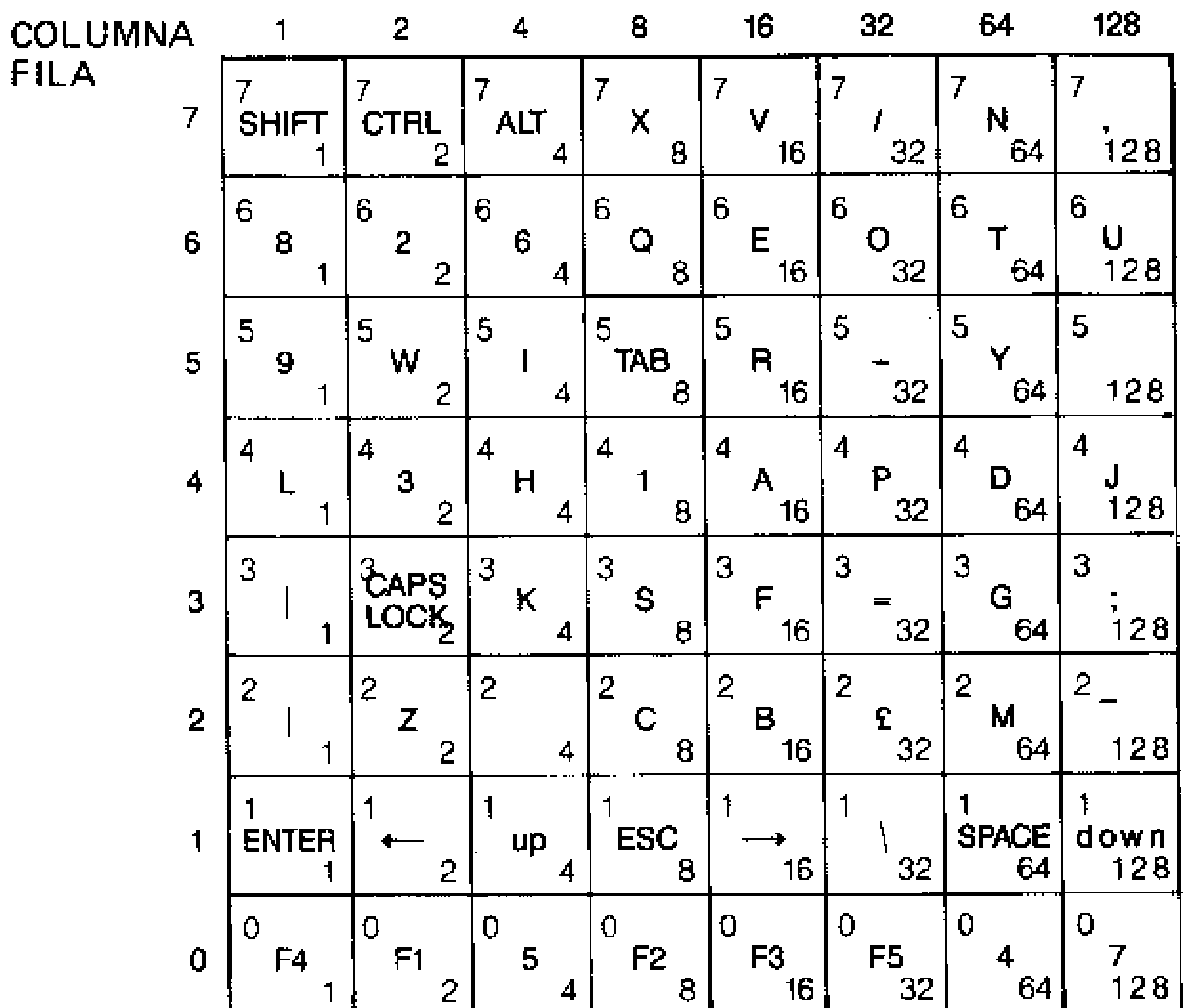
8.6.2. La función KEYROW

El teclado del QL está organizado en filas y la función KEYROW devuelve el número de fila y la codificación de la tecla apretada. La figura 8.2 muestra la tabla de las codificaciones de filas y columnas del teclado.

El formato de esta función es:

KEYROW *(fila)*

donde *fila:* representa la fila elegida y debe ser una expresión-numérica o una constante o una variable de este tipo, cuyo resultado de evaluación esté comprendido dentro del rango de 0 a 7.



FILA \ COLUMNA	1	2	4	8	16	32	64	128
7	SHIFT	CTRL	ALT	X	V	/	N	,
6	8	2	6	Q	E	O	T	U
5	9	W	I	TAB	R	-	Y	
4	L	3	H	1	A	P	D	J
3	|	CAPS LOCK	K	S	F	=	G	;
2	|	Z		C	B	£	M	_
1	ENTER	←	up	ESC	→	\	SPACE	down
0	F4	F1	5	F2	F3	F5	4	7

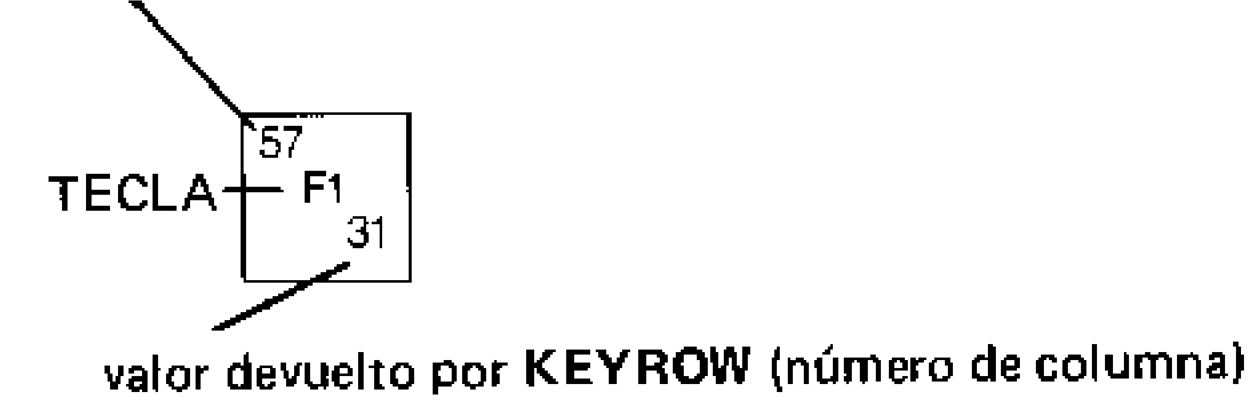


Fig. 8.2.– Codificaciones de la función KEYROW.

Ejemplos:

El siguiente programa ilustra el uso de la función KEYROW.

```
100  REMark    Pulse algunas teclas
110  REPeat bucle
120          CURSOR 0,0
130          FOR fila = 0 TO 7
```



```
140               PRINT fila ! KEYROW (fila); " "
150            END FOR fila
160  END REPeat bucle
```

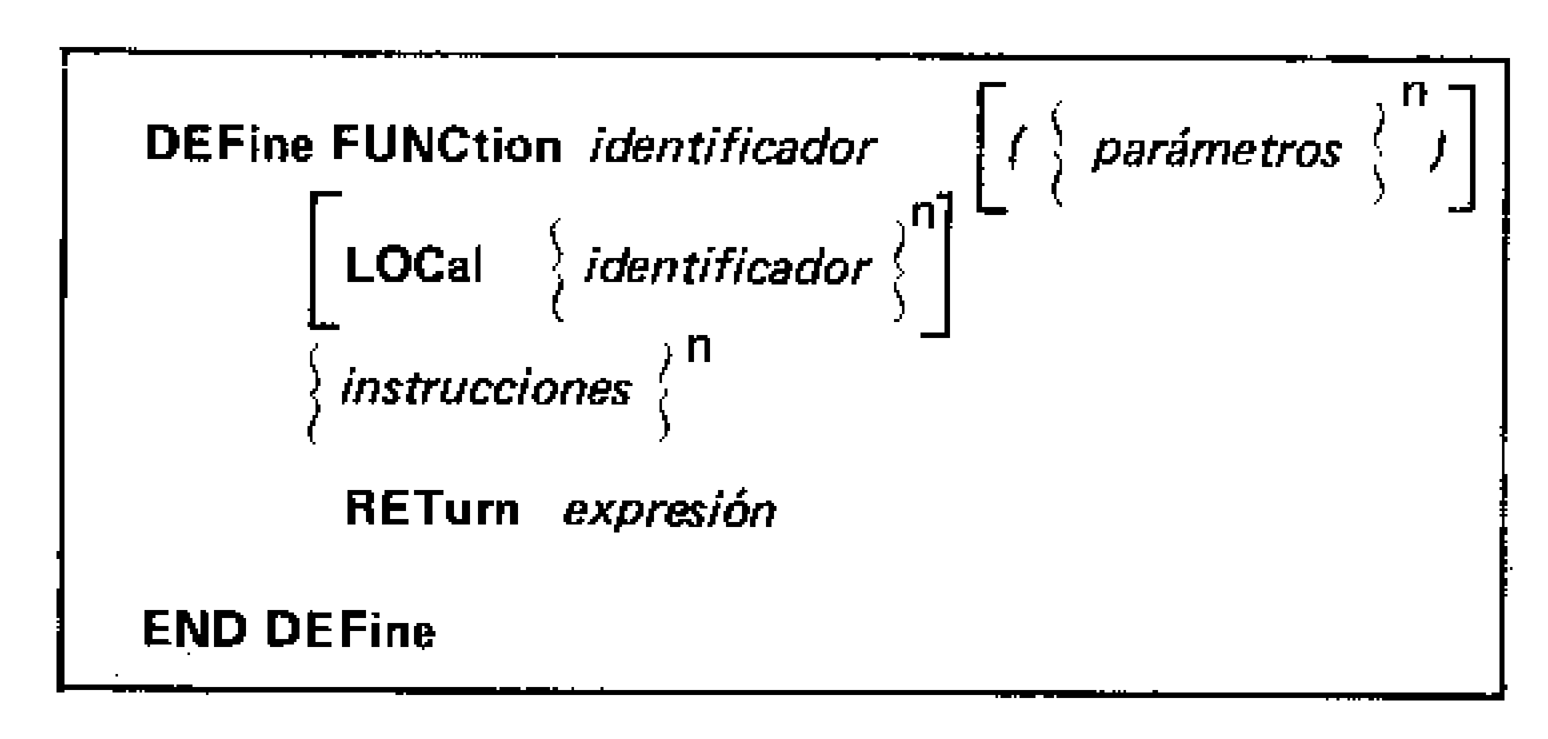
La instrucción CURSOR que aparece en la línea 120 del programa anterior, será estudiada con detalle en el capítulo siguiente. Baste ahora con decir que tal sentencia posiciona el cursor en las coordenadas de la pantalla que se especifican.

8.7. LAS FUNCIONES (FUNCTIONS)

Ya hemos visto en el apartado anterior algunas funciones incorporadas del SuperBASIC que permiten al programador utilizar cierto tipo de facilidades que no tiene que programar, como puedan ser las raices cuadradas, hallar el valor absoluto, las funciones trigonométricas, etc.

No obstante, el SuperBASIC del QL da facilidades al usuario para que éste pueda escribir sus propias funciones con cantidades específicas y que podrá utilizar de forma análoga a las funciones vistas con anterioridad.

El formato general de una función de usuario es:



Una función consta de tres partes claramente diferenciadas:

DEFine FUNCtion *nombre*	CABECERA
cuerpo de la función	instrucciones
RETurn *variable*	DEVOLUCION DE VALOR
END DEFine	FINAL

La *variable* definida en el formato anterior debe ser usada en el cuerpo de la función.

Ejemplo:

Definamos una función que calcule el cubo de un número.

```
50   DEFine FUNction cubo (numero)
60        resultado = numero * numero * numero
70   RETurn resultado
80   END DEFine
```

En el segmento de programa anterior se ha definido una función (línea 50) llamada *cubo* y que posee un único *parámetro de entrada (número)* con el que se opera en el cuerpo de la función (línea 60). La línea 70 contiene la sentencia RETurn que indica que el valor de resultado debe ser *devuelvo* al programa principal. La sentencia END DEFine de la línea 80 da fin a la función como tal.

Este ejemplo es muy sencillo, pero creemos que bastante ilustrativo de cómo deben escribirse funciones en SuperBASIC.

La llamada a una función desde el programa principal puede hacerse en cualquier lugar del mismo, formando parte de una expresión o de una instrucción compleja, de la forma:

nombre-función (parámetros)

Ejemplos:

En el siguiente programa se calculan e imprimen los cubos de los diez primeros números naturales, utilizando la declaración de función anterior.

```
10   REMark  Ejemplo de uso de funcion
20   FOR i = 1 TO 10
30        PRINT i ! cubo (i)
40   END FOR i
70   DEFine FUNction cubo (num)
80        res = num * num * num
90   RETurn res
100  END DEFine
```


Obsérvese en el programa anterior como es en la línea 30 cuando se llama a la función declarada más abajo, mediante la invocación:

```
cubo (i)
```

En ese momento, el valor del *parámetro-actual* (i) se pasa al *parámetro-formal* (num) de la función y se ejecutan las instrucciones de esta función con este valor, devolviéndose como resultado de la función el contenido de la variable *res* mencionada en la sentencia RETurn.

La llamada a la función *cubo* se realizará, por la propia lógica del programa, diez veces según varíe el índice de la instrucción FOR en la que está inmersa.

La figura 8.3 muestra un ejemplo del trasiego de las informaciones entre el programa principal y la función.

Fig. 8.3.— Esquema de intercambio de datos entre el programa principal y una función.

Una variante permitida del uso de la sentencia RETurn es cuando se aplica directamente a una expresión y no a una única variable.

Ejemplo:

```
10   FOR i = 1 TO 10
20        PRINT i ! cubo (i)
30   END FOR i
40   DEFine FUNction cubo (num)
50            RETurn num * num * num
60   END DEFine
```

Como se observa en el programa anterior, se ha aprovechado la instrucción RETurn de la línea 50 para operar con el parámetro-formal *num* y cuyo resultado de evaluación de la expresión es precisamente el valor que se devuelve a la llamada función en el programa principal.

Ejercicio 1

Escribir un programa que use una función y que calcule el factorial de un número.

Recordamos al lector que el factorial de un número n se calcula según la fórmula:

$$n! = n * n\text{-}1 * n\text{-}2 * \ldots * 3 * 2 * 1$$

```
100  REMark ----------------------
110  REMark          Calculo del factorial
120  REMark ----------------------
130  DEFine FUNction fact (k)
140     IF k = 0 OR k = 1 THEN
150        fact = 1
160     ELSE
170        fact = k * fact(k-1)
180     END IF
190     RETurn
200  END DEFine
210  REMark ----------------------
220  INPUT "teclee numero" ! número
230  PRINT "factorial =" ! fact(numero)
240  STOP
```


En este ejemplo puede observarse como existen una llamada a la función "fact" dentro de la propia función. Esto es lo que hemos llamado *recursividad entre funciones y procedimientos* del SuperBASIC.

Ejercicio 2

Escribir un programa con una función que lea 10 parejas de números y que compruebe si dichos valores que representan a "x" y a "y" como puntos del plano interiores al círculo:

$$x^2 + y^2 = 36$$

```
100  REMark ----------------------
110  REMark       Verificacion de puntos
120  REMark ----------------------
130  DEFine FUNction sino (abs, ord)
140     IF abs ** 2 + ord ** 2 <= 36
150        THEN sino = 1
160        ELSE sino = 0
170     END IF
180     RETurn
190  END DEFine
200  REMark ----------------------
210  FOR i = 1 TO 10
220     INPUT "teclee x, y" ! x ! y
230     IF sino (x, y)
240        THEN PRINT "dentro"
250        ELSE PRINT "fuera"
260     END IF
270  END FOR i
280  STOP
```

Ejercicio 3

Escribir un programa que lea una matriz cuadrada de la forma:

$$\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix}$$

y que verifique si es o no simétrica.

Como es sabido, una matriz es simétrica respecto de su diagonal principal si para cualquier elemento a_{ij} se verifica que:

$$a_{ij} = a_{ji}$$

```
100 REMark --------------------
110 REMark        Matriz simetrica
120 REMark --------------------
130 DIMension tabla (10, 10), matriz (10,10)
140 DEFine FUNction simetria(tabla,filas)
150    LOCal i, j
160    simetria = 1
170    FOR i = 1 TO filas - 1
180       FOR j = 1 TO filas
190          IF tabla(i,j) <> tabla(j, i) THEN simetría = 0
200       END FOR j
210    END FOR i
220    RETurn
230 END DEFine
240 REMark --------------------
250 INPUT "dimension?" ! n
260 FOR i = 1 TO n
270    FOR j = 1 TO n
280       INPUT matriz (i, j)
290       IF simetría(matriz,n) THEN PRINT "simetrica"
300                           ELSE PRINT "no sime."
310       END FOR j
320    END FOR i
330 STOP
```

8.8. LOS PROCEDIMIENTOS (PROCEDURES)

Una buena forma de atacar los problemas susceptibles de ser mecanizados por un ordenador es dividiéndolos en tareas o subprogramas más sencillos de abordar individualmente y que definan por sí mismas la identidad de lo que se realiza.

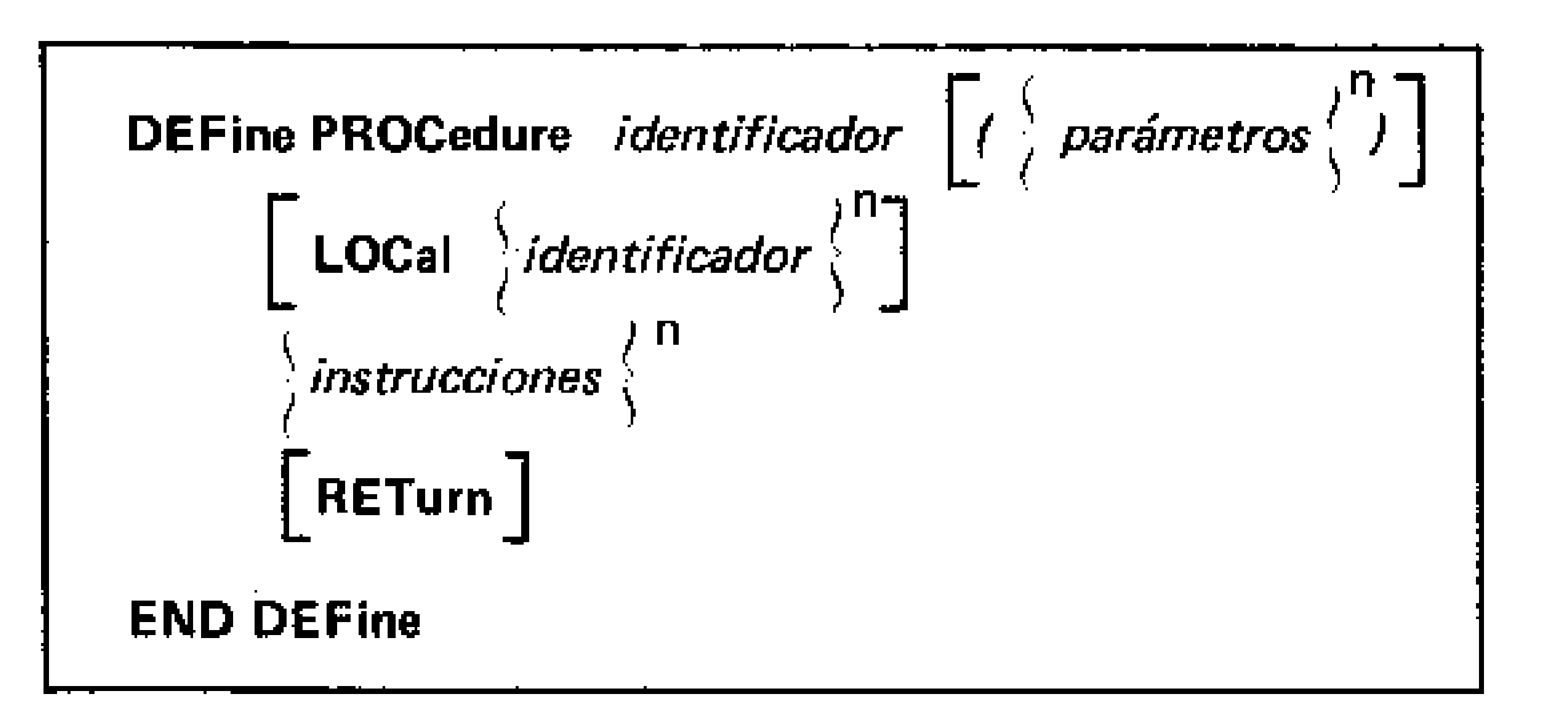
Esta división de los problemas en tareas cada vez más simples se da frecuentemente a lo largo de la vida cotidiana en la jerarquización y racionalización de los trabajos.

Con los programas debe hacerse lo mismo. Un buen programador siempre divide su programa en varios módulos o tareas independientes

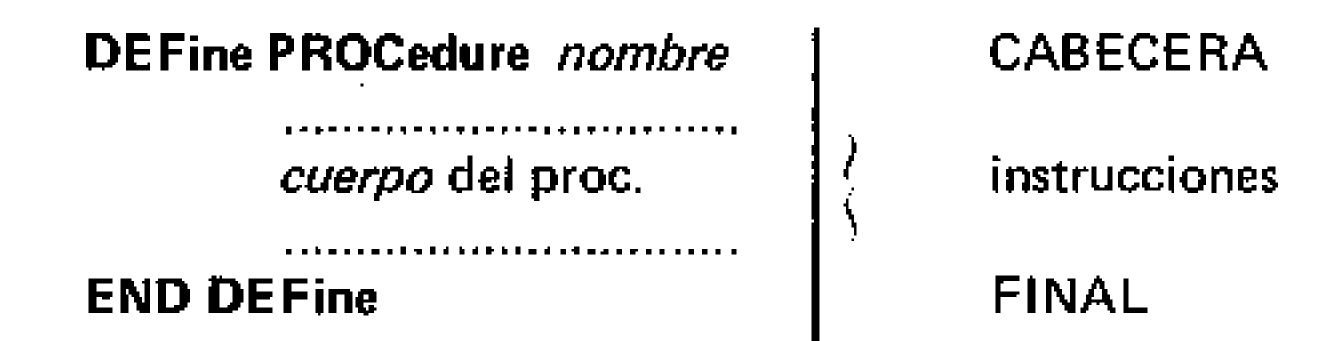
donde cada uno de ellos tiene una misión específica que puede ser atacada por el programador al margen de las demás, y de forma que se plantean ventajas de las que luego hablaremos.

Para definir estas tareas se utilizan en SuperBASIC los procedimientos (procedures).

El formato general de un procedimiento es:



Un procedimiento consta de tres partes claramente diferenciadas:



La *cabecera* indica el comienzo del procedimiento y le da nombre. Este *nombre* debe regirse por las reglas habituales de formación de los identificadores. Acto seguido se escriben las *instrucciones* normales de este procedimiento que serán cualquiera de las del repertorio del Super BASIC para finalizar con la declaración de final de procedimiento.

Un procedimiento deberá comenzar con las palabras DEFine PROCedure y el nombre del procedimiento y terminar con END DEFine.

Ejemplo:

Ilustraremos el uso de los procedimientos con el siguiente ejemplo sencillo.

Se trata de escribir un programa que lea diez parejas de números y que guarde y escriba de cada pareja el que sea mayor.

Planteado así el problema, podemos distinguir en él tres tareas independientes:

1. Lectura de las diez parejas
2. Selección y salvaguarda del mayor de cada una.
3. Impresión del mayor.

Para el almacenamiento de las parejas podremos utilizar matrices y declararlas al comienzo del programa.

Esto, visto así, podría estructurarse con tres procedimientos *lectura*, *selección* y *escritura* dispuestos, por ejemplo, como:

```
1  Declarar (DIM) las matrices
2  lectura
3  FOR 10 parejas
        selección
        escritura
```

El programa que realizará todo esto podría ser:

```
100  Remark -------------------
110  REMark          Clasificacion
120  REMark -------------------
130  DIMension num1(10), num2(10), mayor(10)
140  lectura
150  FOR pareja = 1  TO  10
160       seleccion
170       escritura
180  END FOR pareja
190  REMark -------------------
200  REMark          Procedures
210  REMark -------------------
220  DEFine PROCedure lectura
230          FOR i = 1 TO 10
240             INPUT "pareja?";num1(i);num2(i)
250          END FOR i
260  END DEFine
270  REMark -------------------
280  DEFine PROCedure seleccion
290          FOR i =  1 TO 10
300             IF num1 (i) > num2(i) THEN mayor (i) = num1
310                                  ELSE mayor(i) = num2
320          END FOR i
```



```
330  END DEFine
340  REMark -------------------
350  DEFine PROCedure escritura
360          FOR  i = 1 TO 10
370             PRINT  mayor(i) !
380          END FOR i
390  END DEFine
```

Como acabamos de ver, se han utilizado tres procedimientos en el programa anterior, que poseen cierta peculiaridad: no poseen parámetros.

Un procedimiento puede tener ciertos valores susceptibles de ser cambiados cada vez que se invoca al procedimiento. En este caso, lo único que varía es la *cabecera* del procedimiento que toma la forma:

```
DEFine PROCedure  procedimiento (para1, para2, ..., paran)
```

A estos parámetros escritos en la cabecera de la declaración de un procedimiento se les llama *parámetros formales*.

La forma de llamar a un procedimiento con parámetros es casi análoga.

```
procedimiento p1, p2, ..., pn
```

donde los p1, p2, ..., pn se llaman *parámetros reales*.

Cuando un programa SuperBASIC se encuentra con una llamada a un procedimiento con parámetros, el paso del valor o valores se realiza posicionalmente según se encuentren escritos.

parámetro real 1 → parámetro formal 1
parámetro real 2 → parámetro formal 2
....
parámetro real n → parámetro formal n

Ejemplo:

Definamos un procedimiento que dibuje una circunferencia de origen y radio aleatorios.

Este procedimiento podría ser: (1)

(1) Nota: La instrucción CIRCLE será estudiada con detalle en el capítulo siguiente. Baste ahora mencionar que su misión consiste en dibujar una circunferencia de origen (x, y) y radio r.

```
10   DEFine PROCedure circunferencia (x, y, r)
20             CIRCLE x, y, r
30   END DEFine
```

Como se ve en el procedimiento anterior se utilizan tres parámetros formales llamados x, y, r que deberán ser "llenados" en las invocaciones al procedimiento.

Confeccionamos ahora un programa general que dibuje aleatoriamente diez circunferencias en la pantalla y de radio comprendido entre 20 y 40.

```
10   DEFine PROCedure circunferencia (x, y, r)
20         CIRCLE, x, y, r
30   END DEFine
40   REMark----------------------
50   FOR i = 1 TO 10
60      abscisa = RND (0 TO 100)
70      ordenada = RND (0 TO 100)
80      radio = RND (20 TO 40)
90      circunferencia abcisa, ordenada, radio
100  END FOR i
```

Obsérvese como en la línea 90 del programa anterior se llama al procedimiento *circunferencia* con los valores que posee en cada momento las variables abcisa, ordenada y radio. Naturalmente, cada vez que se ejecute este programa se obtendrán resultados diferentes.

Hagamos ahora un programa que realice lo mismo pero con cuadrados en vez de circunferencias. Intente el lector pergeñar la solución antes de observar la propuesta (1).

```
100  DEFine PROCedure cuadrado (x, y, ℓ)
110            LINE x, y TO x, y + 1
120            LINE TO x + 1, y+ 1
130            LINE TO x + 1, y
140            LINE TO x, y
150  END DEFine
160  REMark-----------------
170  FOR i = 1  TO 10
```

(1) Nota: La instrucción LINE que se muestra en este programa también será estudiada con detalle en el capítulo siguiente. Unicamente diremos ahora que su función se limita a dibujar una línea recta entre dos puntos dados.


```
180           abcisa = RND (0 TO 100)
190           ordenada = RND (0 TO 100)
200           lado =  RND (20 TO  40)
210           cuadrado  abcisa, ordenada, lado
220  END FOR i
```

Hasta ahora hemos visto cómo se pasaban *parámetros-valor* de la llamada de un procedimiento al propio cuerpo del procedimiento, esto es, los procedimientos no devolvían a la instrucción de llamada ningún valor. A esta segunda forma de paso de parámetros se la conoce como *parámetros-variable.*

Ejemplo:

```
10   REMark ------------------
20   REMark        Parametros-variable
30   REMark ------------------
40   circun 5, long1
50   circun 7, long2
60   PRINT "la suma es = " ! long1 + long2
70   DEFine PROCedure circun (radio, longitud)
80             longitud = 2 * PI * radio
90   END DEFine
```

En este programa puede observarse cómo la llamada al procedimiento *circun* (que calcula la longitud de una circunferencia dado su radio) se realiza con dos parámetros. El primero de ellos pasa a ocupar el lugar de la variable *radio* en el procedimiento y el resultado obtenido en *longitud* se devolverá primero a la variable *long1* (línea 40) y después a *long2* (línea 50).

La figura 8.4 muestra un ejemplo del paso de informaciones entre el programa principal y el procedimiento.

8.9. VARIABLES GLOBALES Y VARIABLES LOCALES

Cualquier variable que sea mencionada como un parámetro formal dentro de una declaración de una función o de un procedimiento es, por definición, *local;* es decir, de uso exclusivo para la función o procedimiento donde está definida.

Sin embargo, todas las restantes variables que pueden usarse dentro de una función o procedimiento son *globales*, es decir, pueden usarse

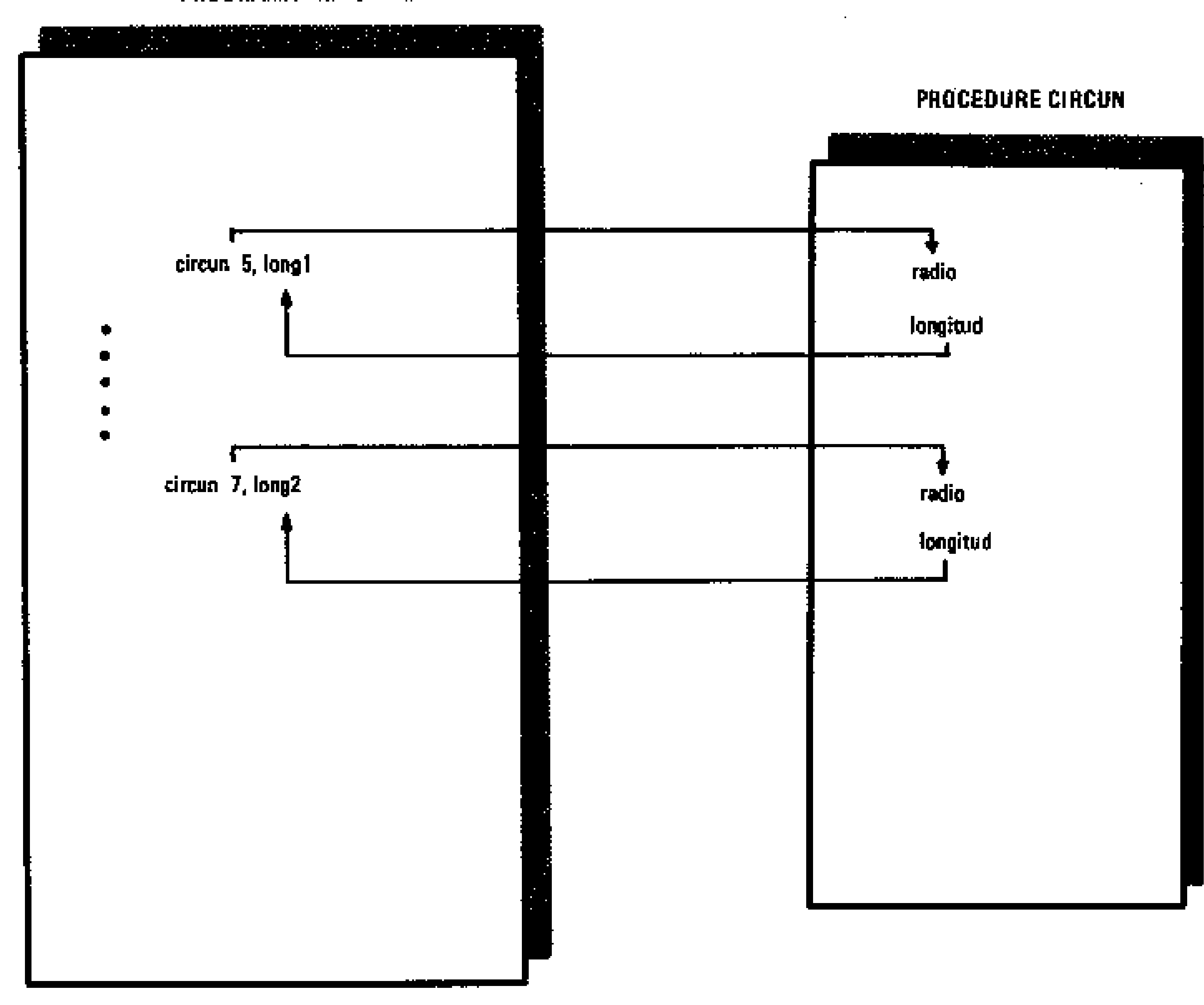


Fig. 8-4.— Intercambio de informaciones entre el programa principal y un procedimiento.

indistintamente dentro y fuera del cuerpo del procedimiento o función de que se trate.

En SuperBASIC, en común con otros lenguajes como PASCAL, etc, pueden declararse variables locales dentro de una función o procedimiento de la forma ya vista en los formatos anteriores:

LOCal *variable, variable, ...*

Ejemplos:

```
80   LOCal a1, b, d (20, 10)
100  LOCal variable_temporal
```


Ejemplo:

Sea el siguiente procedimiento:

```
50   DEFine PROCedure pot (num)
60   LOCal i
70   FOR i = 1 TO 100
80        PRINT i ! i * i ! i * i * i
90   END FOR i
100  END DEFine
```

En el procedimiento anterior, la variable *num* como se trata de un parámetro-formal del procedimiento se toma como una *variable local* exclusiva para dicho procedimiento. Si no existiera la declaración de LOCal de la variable *i* de la línea 60, esta variable sería tomada en cuenta como global, pudiéndose utilizar en todo el programa. Como existe la declaración LOCal i, entonces esta variable es tomada como local exclusivamente para el cuerpo del procedimiento en la que está definida.

Cualquier otra referencia a la variable *i* en cualquier otro lugar del programa será tomada como otra variable (con el mismo nombre) y no podrá existir nunca problemas de interdependencia entre ellas.

Ejemplo:

Considerese el siguiente programa:

```
10   REMark Amplitud de las variables
20   numero = 1
30   prueba
40   DEFine PROCedure prueba
50        LOCal numero
60        numero = 2
70        PRINT numero
80        intento
90   END DEFine
100  DEFine PROCedure intento
110       PRINT numero
120  END DEFine
```

¿Qué valores serán impresos?

En la línea 20 se utiliza la variable *número* como una variable global para todo el programa. Por otro lado en la línea 50 se declara a la variable *número* como *local* al procedimiento *prueba* y serán tomados como *distintas* en este momento.

Continuando con el proceso, también se hace mención a la variable *número* en la línea 110.

Cualquier variable que sea local de un procedimiento será la misma variable dentro de un segundo procedimiento que sea llamado por el primero.

Como colofón mencionaremos algunas de las ventajas que ofrece la modularización de los programas en forma de procedimientos.

1. Puede usarse el mismo segmento de programa varias veces dentro del mismo o bien dentro de otro programa.
2. Pueden subdividirse las tareas en subtareas y escribir un procedimiento para cada una de las subtareas. Naturalmente, hacer esto así supone una ayuda considerable a la hora del análisis y del diseño.
3. Los procedimientos pueden ser verificados separadamente, lo que conlleva una ayuda inestimable para el momento de la puesta a punto y la depuración.
4. La utilización de procedimientos con fines específicos ayudan notablemente a hacer un programa más comprensible y legible.

Como muestra de las ventajas de la utilización de los procedimientos frente a las tradicionales subrutinas llamadas con GOSUB, realizaremos ahora un experimento con dos programas: uno con GOSUB y el otro con un procedimiento, que realizan la misma función y que es dibujar un cuadrado de color y dimensiones variables y localización variable (1).

Opción GOSUB

```
100  color = 4
110  fondo = 2
120  abscisa = 200: ordenada = 100
130  lado = 50
140  GOSUB 160
```

(1) Nota: Las instrucciones PAPER y BLOCK serán estudiadas con detalle en el capítulo siguiente.


```
150  STOP
160  REMark Subrutina del cuadrado
170  PAPER fondo : CLS
180  BLOCK lado, lado, abscisa, ordenada, color
190  RETURN
```

Opción PROCEDURE

```
100  DEFine PROCedure cuadrado (color, lado, abscisa, ordenada, fondo)
110            PAPER fondo: CLS
120            BLOCK lado, lado, abscisa, ordenada, color
130  END DEFine
140  cuadrado 4,50,20,100,2
```

Como puede observarse, en el primer caso, los valores de color, abscisa, ordenada, lado y fondo son previamente «cargados» con una sentencia de asignación, mientras que en el segundo caso, esto no es necesario, dado que en la llamada al procedimiento ya se especifican estos valores. El uso del procedimiento permite una mayor claridad a la hora de la depuración del programa y de las pruebas.

Naturalmente, cuanto más largo es un programa, mayores son las ventajas que se desprenden de la utilización de procedimientos, que sin duda acortarán el número de líneas necesarias, caso que ha sido demostrado palpablemente en el ejemplo anterior.

9

Los gráficos

9.1. INTRODUCCION

En este capítulo se describen las posibilidades en la realización de gráficos del QL. Se estudiarán con detalle cada uno de los sistemas de representación, así como las instrucciones para el dibujo de gráficos en la pantalla conectada al ordenador y se ilustrará todo ello con ejemplos y programas SuperBASIC para gráficos.

La forma de representación de gráficos puede realizarse en el QL según dos sistemas algo diferentes.

El primero de ellos es el llamado *sistema gráfico de coordenadas* que utilizan los procedimientos gráficos empleando formas relativas al *origen gráfico*. La figura 9-1 muestra un esquema gráfico de tal sistema.

Como puede observarse en la figura 9-1, el origen de este sistema gráfico se encuentra en la esquina inferior izquierda de la pantalla. El cursor puede desplazarse en todo el contexto del gráfico, de forma que sus sucesivas posiciones pueden ser *relativas* respecto al punto anterior o *absolutas* respecto del origen de coordenadas.

En este sistema el *factor de escala* del eje de coordenadas (eje y) es de 100 si no se especifica otra cosa. No obstante, y como ya veremos, este parámetro puede alterarse con la instrucción SCALE.

Hablaremos en muchas ocasiones del concepto de *ventana (window)* y podemos definir a las ventanas como porciones separadas e independientes de espacio gráfico donde pueden dibujarse figuras. Cada una de estas ventanas podrá ser *creada* con un número (p. ej. # 5) al que llamaremos de ahora en adelante *canal* y que aparecerán en muchas de las definiciones de formatos de las instrucciones gráficas.

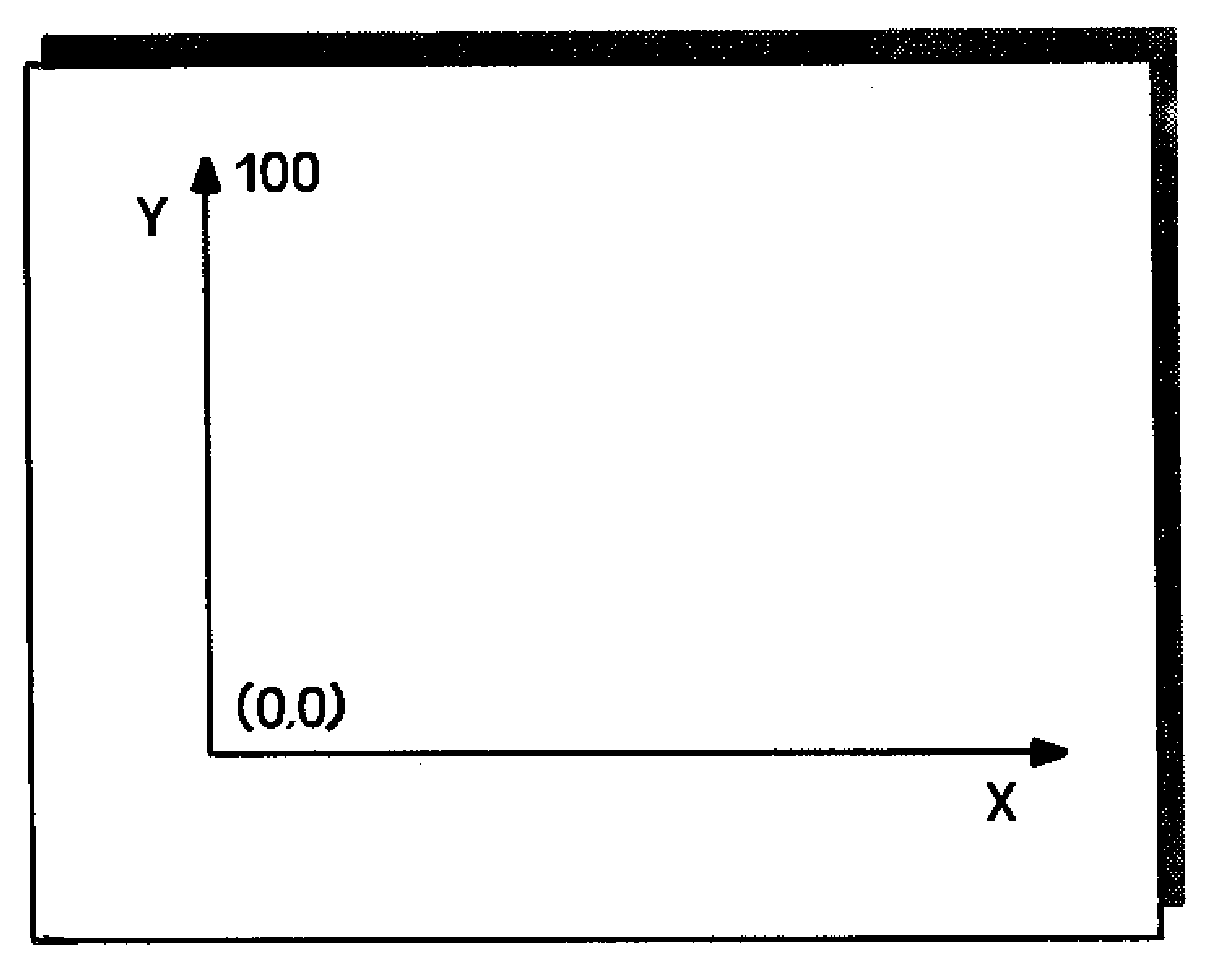


Fig. 9.1.— Sistema gráfico de coordenadas.

El segundo sistema de representación de gráficos es el llamado *sistema de coordenadas de pixels*. Un *pixel*(1) es la mínima unidad susceptible de ser representada en una pantalla y puede asimilarse con pequeños puntos donde el número de ellos depende del modo de *resolución* utilizado. En el QL, como ya mencionábamos en el capítulo 0, existen dos posibles modos de resolución: *alta resolución* (512 x 256 pixels) y *resolución media* (256 x 256 pixels).

Para la definición de las posiciones relativas o absolutas de ventanas de trabajo, rectángulos, etc., se utiliza en SuperBASIC el concepto de sistema de coordenadas de los pixels.

El origen de coordenadas de los pixels, tal y como se muestra en la figura 9-2 se sitúa en la esquina superior izquierda de la pantalla.

(1) Pixel es la abreviatura de "picture element".

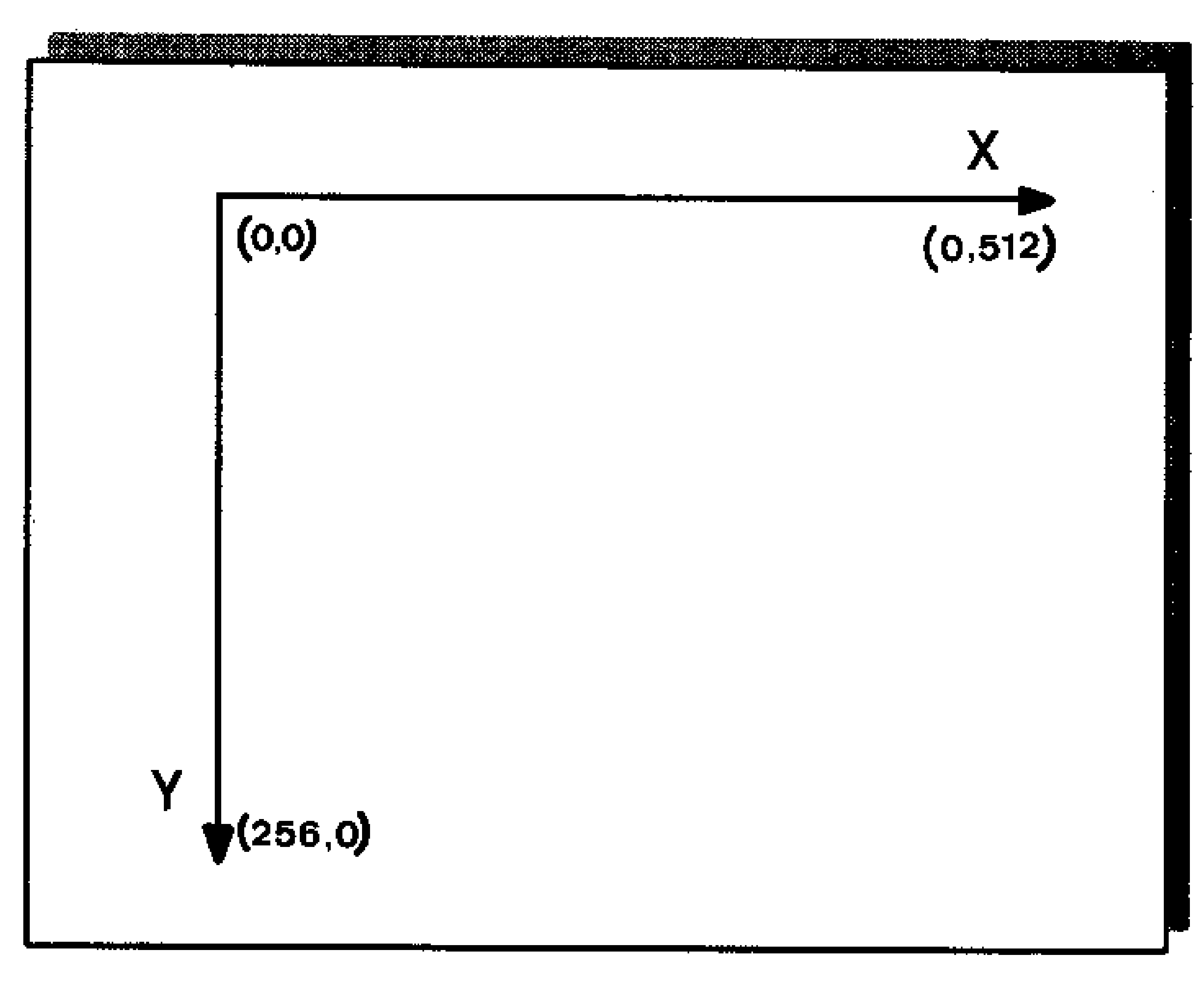


Fig. 9.2.– Sistema de coordenadas de pixels.

Como puede observarse en la figura anterior, este sistema de coordenadas asume que la pantalla se encuentra en el modo de alta resolución.

Como hemos dicho, el SuperBASIC del QL dispone de instrucciones adecuadas y potentes para el manejo de gráficos en la pantalla conectada al ordenador. Veámoslas separadamente.

9.2. LA INSTRUCCION SCALE

Con la sentencia SCALE puede alterarse el factor de coordenadas utilizado en el sistema gráfico.

El formato de la instrucción es:

SCALE [*canal,*] *escala, x, y*

donde *escala:* es el factor (número de pixels) que se fijan en el eje de ordenadas para realizar dibujos. El factor de escala por defecto es 100.

x, y: Son las coordenadas del origen del nuevo sistema gráfico.

La instrucción SCALE puede aplicarse, como se desprende de la observación del formato de la sentencia, sobre cualquier canal (ventana) previamente definida.

Ejemplos:

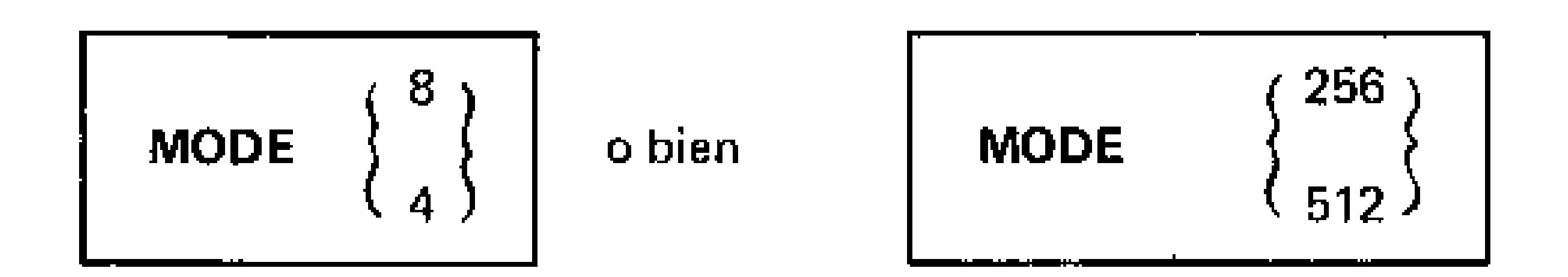
20 **SCALE** 10,0,0
fija una escala de 10 unidades y con origen en el punto (0,0)

40 **SCALE** 75,20,25
fija una escala de 75 unidades y con origen en el punto (20,25)

9.3. LA INSTRUCCION MODE

Esta instrucción se utiliza para definir el modo de resolución de la pantalla o monitor conectada al QL.

El formato de la instrucción es el siguiente:



Con mode 256 (equivalente a MODE 8) se obtiene la pantalla con resolución media (256 puntos) y se tiene la posibilidad de utilizar de entre 8 colores diferentes. Análogamente con MODE 512 (equivalente a MODE 4) se obtiene alta resolución en pantalla (512 puntos) y se tiene la posibilidad de utilizar de entre 4 colores diferentes.

La tabla de la figura 9-3 muestra un cuadro de utilización de los colores (con sus códigos respectivos) para cada uno de los dos modos descritos anteriores.

CODIGO	COLOR	
	MODE 8	MODE 4
0	negro	negro
1	azul	negro
2	rojo	rojo
3	magenta	rojo
4	verde	verde
5	cyan	verde
6	amarillo	blanco
7	blanco	blanco

Fig. 9.3.— Cuadro de colores para los dos modos de resolución.

9.4. LA INSTRUCCION INK

Esta instrucción se utiliza para definir el color que será utilizado en el dibujo de gráficos desde el mismo momento de la ejecución de la instrucción y hasta que se encuentre otra sentencia INK que anule a la anterior.

El formato de la instrucción es el siguiente:

INK [*canal,*] *color*

La instrucción INK posee además algunas características importantes. Pueden construirse patrones coloreados. La figura 9-4 muestra un cuadro de los posibles patrones que pueden conseguirse.

CODIGO	NOMBRE	PATRON
0	pixel sencillo de contraste	
1	patrón horizontal	
2	patrón vertical	
3	patrón tablero	

Fig. 9.4.— Patrones gráficos.

Puede utilizarse la instrucción INK para construir estos patrones según el formato:

INK *color, contraste, código-patrón*

donde *color* y *constante* se corresponden con los colores que se muestran en la figura 9-3.

Ejemplos:

La instrucción INK 2,4,1 producirá un patrón de la forma:

rojo	rojo
verde	verde

La instrucción INK 2,4,0 producirá un patrón de la forma:

rojo	verde
rojo	rojo

9.5. LA INSTRUCCION PAPER

Esta instrucción se utiliza para definir el color de fondo que poseerá la pantalla con el objeto de realizar sobre ella los más variados gráficos.

El formato de la instrucción es:

PAPER [*canal,*] *color*

donde *color* será uno cualquiera de los mencionados en la tabla de la figura 9-3.

Obsérvese, como siempre, que en MODO 8 se podrán utilizar hasta ocho colores, mientras que en MODO 4 sólo estarán disponibles cuatro.

Al igual que lo que sucede con la instrucción INK, puede utilizarse la instrucción PAPER para obtener uno de los patrones vistos en la figura 9-4.

PAPER *color,contraste,patrón*

Ejemplos:

PAPER 2 (fondo de color rojo)
PAPER 2,4,1 (patron vertical rojo-verde)

9.6. LA INSTRUCCION RECOL

Con esta instrucción pueden trocarse los colores de todos los pixels visualizados en la pantalla o en el canal (ventana) especificada.

El formato de la sentencia es:

RECOL [*canal,*] *c0,c1,c2,c3,c4,c5,c6,c7*

donde *c0:* representará el código del color que sustituirá a todos los pixels de color *negro.*
c1: Idem para *azul.*
c2: Idem para el *rojo.*
c3: Idem para *magenta.*
c4: Idem para *verde.*
c5: Idem para *cyan.*
c6: Idem para *amarillo.*
c7: Idem para *blanco.*

Ejemplo:

```
20   RECOL 2,3,4,5,6,7,1,0
cambiará los colores:
  azul pasará a rojo
  rojo pasará a magenta
  magenta pasará a verde, etc.
```

9.7. LA INSTRUCCION POINT

Esta instrucción dibuja un punto en unas determinadas coordenadas de la pantalla, utilizando el sistema gráfico de coordenadas.

El formato de la instrucción es:

```
{ POINT   }
{         }  [canal,]  { x,y }ⁿ
{ POINT_R }
```

donde *x,y:* representan las coordenadas del punto elegido y pueden ser constantes, variables o incluso expresiones.

Ejemplo:

> 30 **POINT** 60,50
> dibujará un punto (con el color de tinta actual) en las coordenadas (60,50) de la ventana o pantalla en curso.

Existe también la variante relativa de esta sentencia que es POINT_R que hace referencia al punto en el que se encuentre el cursor y no al origen de coordenadas del sistema gráfico.

Ejemplo:

> 40 **POINT_R** 20,30
> dibujara un punto en las coordenadas (20,30) respecto de la posicion del cursor en ese momento.

9.8. LA INSTRUCCION LINE

Esta instrucción dibuja una línea recta en la pantalla desde las coordenadas que se mencionan correspondientes al punto origen (x, y) de la línea hasta las coordenadas de destino (x', y').

El formato de la instrucción es:

```
                        { x,y TO x',y' [{ TO x,y }] }
{ LINE   }              {                           }
{ LINE_R }   [canal,]   { TO x,y [{ TO x,y }]       }
                        {                           }
                        { x, y                      }
```


Tanto las *"x"* como las *"y"* anteriores pueden ser expresiones numéricas o constantes o variables de este tipo.

Cuando se utiliza el formato superior de la instrucción se obtiene una línea desde las coordenadas origen (x, y) hasta las coordenadas destino (x', y'). Cuando se utiliza el formato de la línea intermedia se dibujará una recta desde el punto *actual* que ocupe el cursor en la pantalla hasta las coordenadas destino (x, y).

Cuando se utiliza el formato inferior, el cursor se moverá al punto cuyas coordenadas se especifican sin que se dibuje ninguna línea.

Ejemplo:

El programa siguiente dibuja un cuadrado con el vértice inferior izquierdo en el punto de coordenadas (10,10) y con un lado de 50.

```
10   PAPER 7
20   INK RND (5)
30   LINE 10,10 TO 10,60
40   LINE TO 60,60
50   LINE TO 60,10
60   LINE TO 10,10
```

Obsérvese en este programa como se ha definido con la instrucción PAPER el color de fondo de la pantalla y como con la instrucción INK se ha escogido un número aleatorio entero comprendido entre 0 y 5 de forma que cada vez que se ejecute este programa se obtendrá un cuadrado de un color distinto.

Cuando se utiliza la instrucción LINE anterior deben definirse como hemos visto, las coordenadas de origen y de destino de la línea. No obstante, a veces es más cómodo definir un punto de referencia que no sea precisamente el origen de coordenadas y a partir de él, dibujar las figuras que deseemos. Para ello se utilizan conjuntamente las instrucciones POINT y LINE_R.

Ejemplo:

Estos dos grupos de instrucciones son equivalentes:

```
20   LINE 60,50 TO 70,50 TO 70,60 TO 60,60 TO 60,50
y
10   POINT 60,50
20   LINE_R 0,0 TO 10,0 TO 0,10 TO -10,0 TO 0,-10
```

Ambos grupos de instrucciones dibujan un rectángulo de origen en las coordenadas (60,50).

Obsérvese que en la segunda alternativa se ha señalado primeramente con la instrucción POINT el origen de la línea relativa (LINE_R) que se dibujará a continuación.

9.9. LA INSTRUCCION BORDER

En SuperBASIC, aquella zona de la pantalla que está disponible en un instante para dibujar se la denomina ventana *(window)* y está delimitada por un rectángulo.

La instrucción BORDER se utiliza para dibujar un marco que delimite la ventana de trabajo.

El formato de la instrucción es:

BORDER [*canal,*] *ancho* [*, color*]

donde *ancho* representa una cantidad indicativa del número de pixels (elementos básicos o puntos) que poseerá dicho borde y *color* será uno cualquiera de los vistos en la tabla de la figura 9.3.

Ejemplo:

BORDER 5,2
dibujara un borde para la ventana de trabajo de
5 pixels de ancho y de color rojo (código 2)

Como en muchas otras sentencias puede especificarse una instrucción BORDER para un canal o ventana concreto mencionando dicho número de canal en la propia sentencia:

```
BORDER #8,5,2
```

El siguiente programa ejemplo ilustra el uso de la sentencia BORDER con diferentes tipos de colores destellantes en la pantalla.

Ejemplo:

```
100  FOR a = 85 TO 2 STEP -2
110       BORDER a, RND (0 TO 7)
120       PAUSE 10
130  END FOR a
```

9.10. LA INSTRUCCION BLOCK

Cuando se ejecuta en SuperBASIC una instrucción de este tipo, se dibuja en la pantalla un rectángulo cuyas coordenadas del vértice inferior izquierdo y las medidas de sus dos lados deben ser especificadas en la propia instrucción.

El formato de la sentencia es:

BLOCK [*canal,*] *ancho,alto,abscisa,ordenada,color*

donde *ancho y alto*: son las medidas que poseerán los lados del rectángulo.
abscisa y ordenada: son las coordenadas del vértice inferior izquierdo.
color: es cualquiera de los mencionados en la tabla de la figura 9-3.

Ejemplo:

BLOCK 50,20,10,10,2
dibujara un rectángulo de color rojo como
muestra la figura 9-5.

Naturalmente, puede añadirse el código del canal o de la ventana (window) adecuada allí donde se quiera utilizar.

```
BLOCK #7,50,20,10,10,2
```

9.11. LA INSTRUCCION CIRCLE

Con el SuperBASIC del QL también pueden dibujarse *circunferencias* mediante el uso de la instrucción CIRCLE.

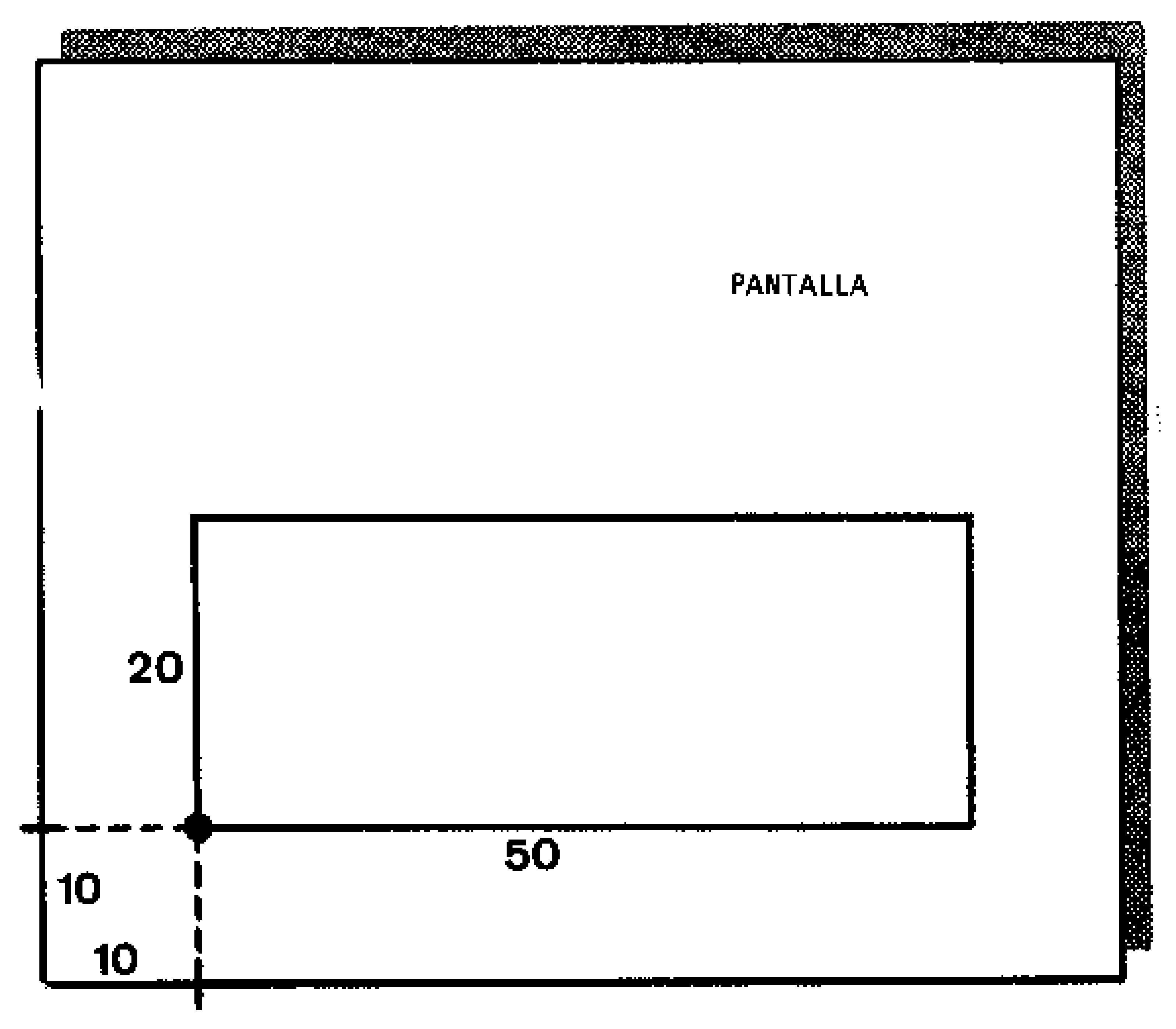


Fig. 9.5.— Uso de la instrucción BLOCK.

En su forma más sencilla, el formato de la instrucción es:

CIRCLE [*canal,*] *abscisa, ordenada, radio*

donde *abscisa y ordenada:* son las coordenadas del origen o centro de la circunferencia y
radio: es la longitud del radio de la circunferencia a dibujar.

Ejemplo:

Sea la instrucción:

20 **CIRCLE** 50,60,40
dibujara una circunferencia tal y como se muestra en la figura 9-6.

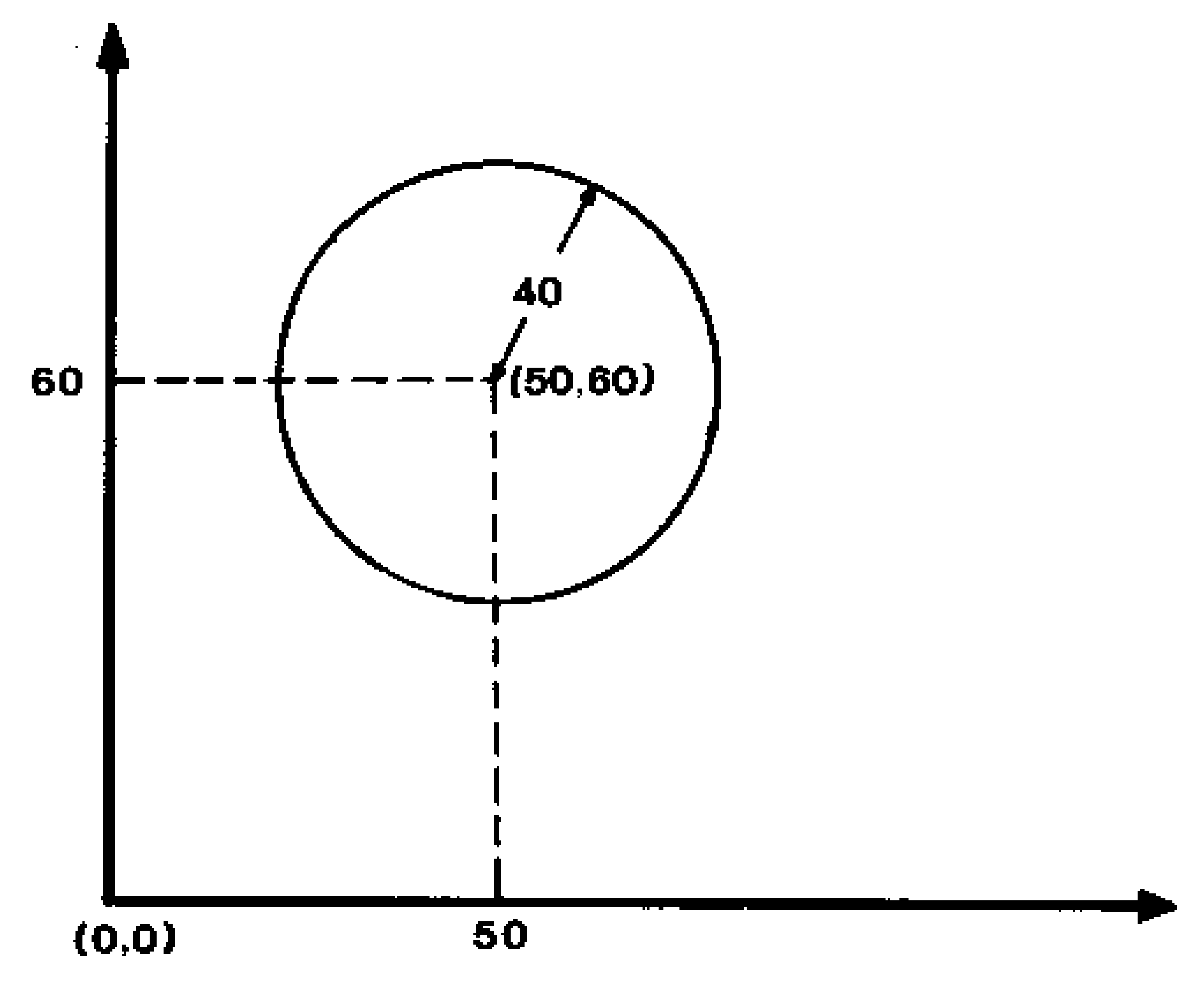


Fig. 9.6.— Ejemplo de representación de una circunferencia mediante la instrucción CIRCLE.

Mediante la instrucción CIRCLE también pueden dibujarse *elipses* sin más que añadir ciertos parámetros a la propia instrucción.

Este formato más complejo es:

CIRCLE [*canal,*] *abscisa, ordenada, semi-eje, excen, orienta*

donde *abscisa y ordenada:* son las coordenadas del centro de la elipse.
semi-eje: es la distancia de la medida del semi-eje mayor de la elipse.
excen: es un número que representa la excentricidad (mayor o menor abertura) de la elipse.
orienta: representa la orientación respecto del eje de ordenadas (expresada en radianes).

Ejemplos:

La instrucción:

20 **CIRCLE** 50,50,40,5,0
dibujara una elipse de la forma mostrada en la figura 9-7.

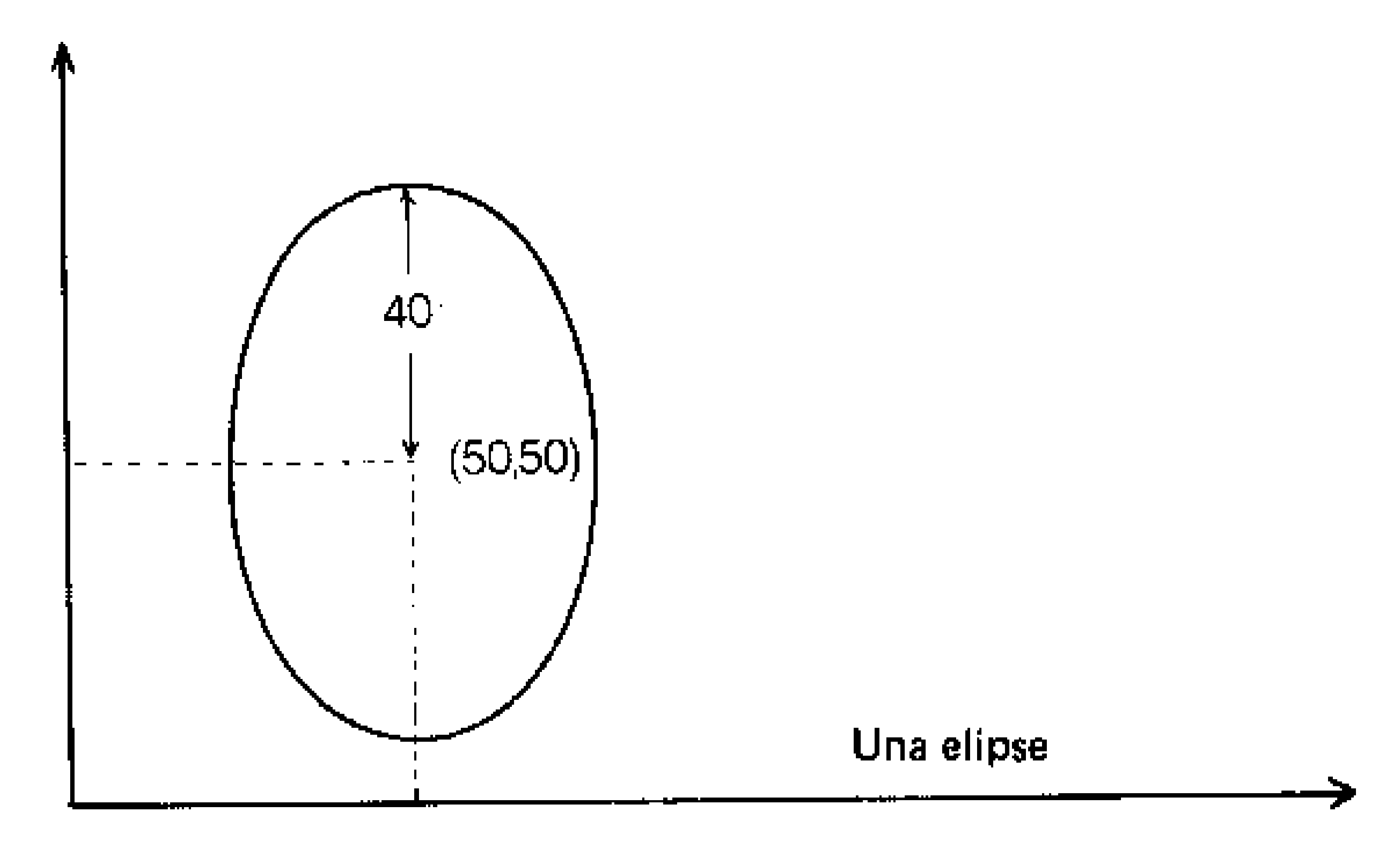


Fig. 9.7.— Ejemplo de representación de una elipse mediante la instrucción CIRCLE.

La instrucción:

20 CIRCLE 50,50,40,.5,1
dibujara una elipse de la forma mostrada en la figura 9-8.

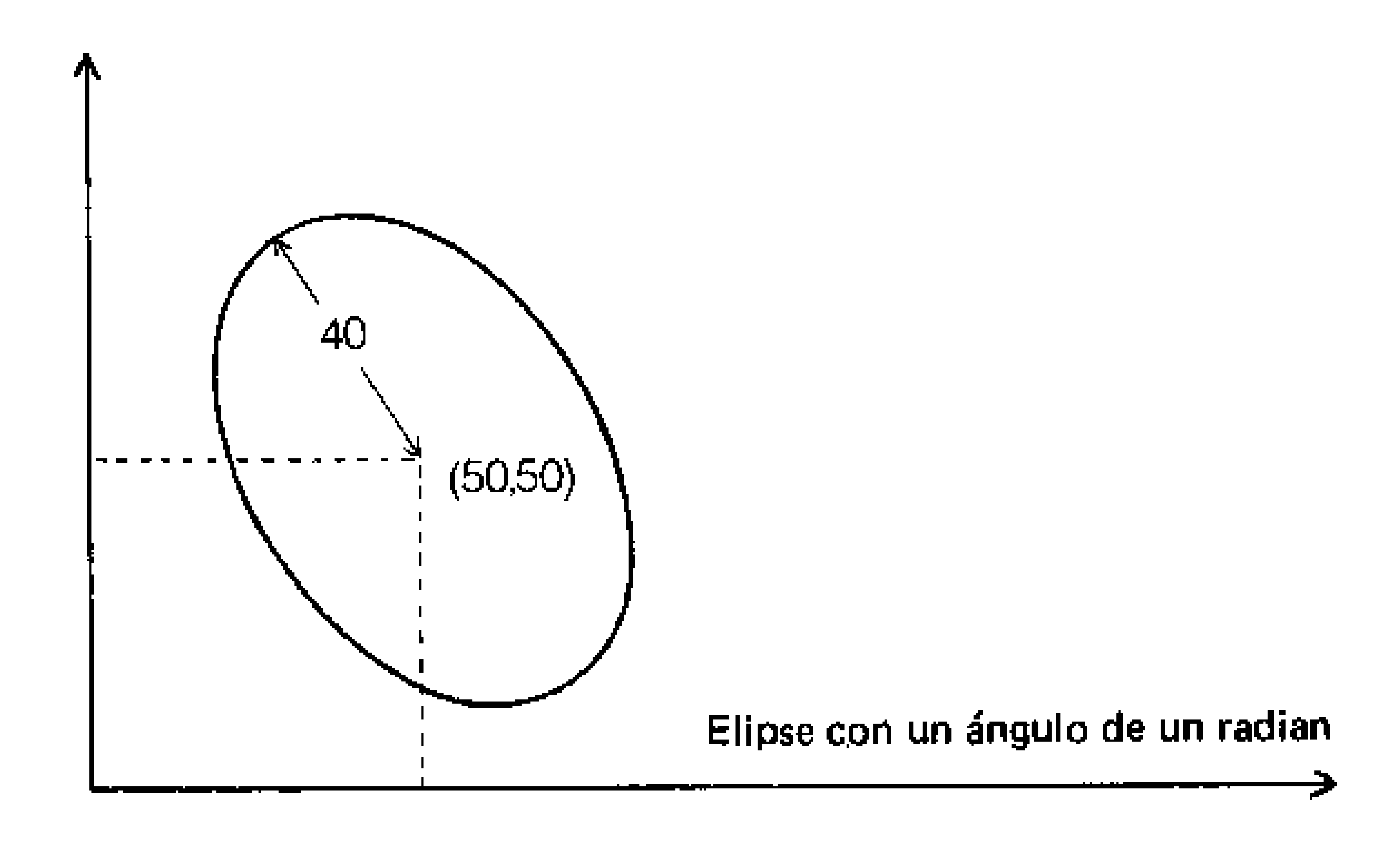


Fig. 9.8.— Elipse con orientación.

El siguiente programa dibujará una serie de elipses, todas con el mismo centro y con una orientación distinta.

```
10   FOR orientacion = 0 TO 2 * PI STEP PI /6
20        CIRCLE 50,50,40,.5, orientacion
30   END FOR orientacion
```

Una instrucción CIRCLE también puede hacer referencia a un punto relativo de la pantalla que ha sido usado o mencionado con anterioridad mediante la variante

CIRCLE_R

y con la utilización de los mismos parámetros anteriores.

Naturalmente, pueden mencionarse en la propia instrucción el canal o ventana (window) donde quiera que se dibuje el círculo.

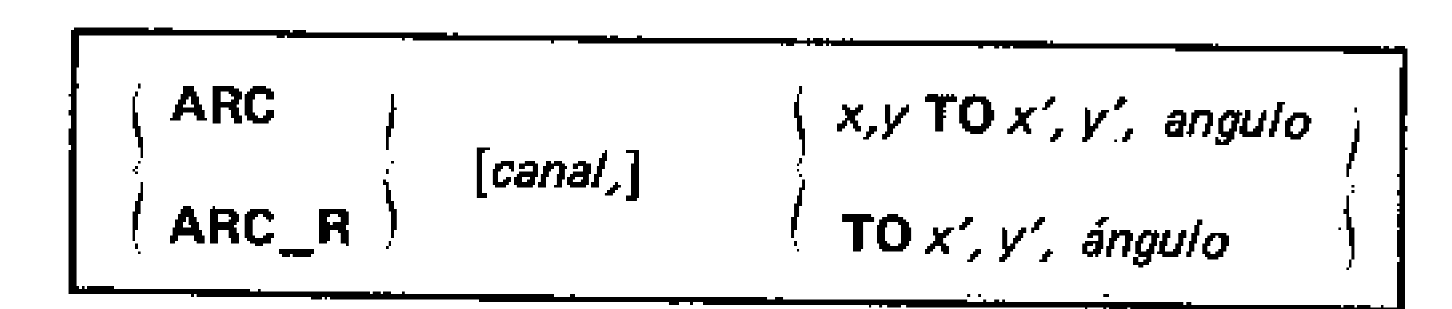
CIRCLE #6,20,30,10
respecto de las coordenadas absolutas (20, 30)

o bien

CIRCLE_R #6,20,30,10
respecto del último punto utilizado.

9.12. LA INSTRUCCION ARC

En SuperBASIC pueden dibujarse también arcos de circunferencias. La instrucción ARC posee el siguiente formato:



donde *x, y:* son las coordenadas del punto origen del arco.
x', y': son las coordenadas del punto de destino del arco.
ángulo: es la medida de la curvatura del arco a dibujar entre los puntos anteriores. Debe ser una cantidad expresada en radianes.

Ejemplo:

La figura 9-9 muestra el resultado de la ejecución de la instrucción ejemplo:

```
20    ARC 10,50 TO 50,90,1
```

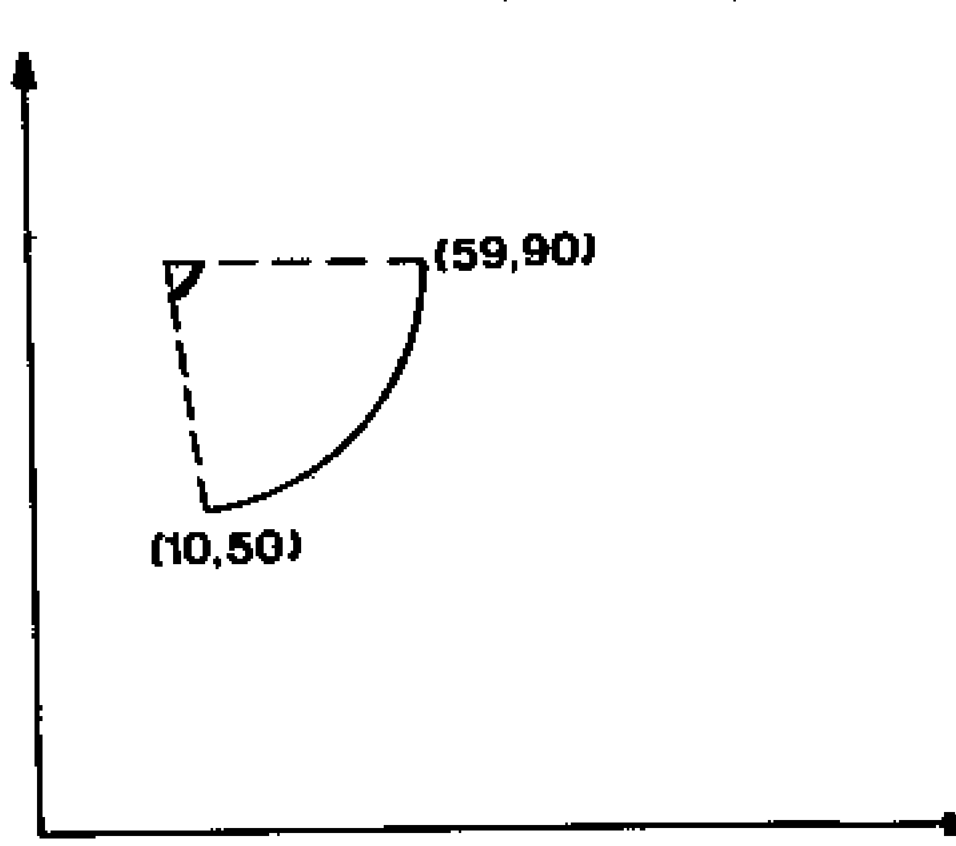


Fig. 9.9.— Ejemplo de representación mediante la instrucción ARC.

La instrucción ARC también puede utilizarse no desde el origen de coordenadas sino tomando como referencia el último punto utilizado, mediante la variante mostrada en el formato anterior.

ARC_R TO *x', y', curvatura*

donde *x', y':* es la coordenada del punto final del arco y
curvatura: es la del arco expresada en radianes.

Ejemplo:

ARC_R 50,60, **PI**/2
que dibujará un arco desde el último punto referido hasta el punto de coordenadas (50,60) con una curvatura de $\pi/2$ radianes.

Tanto la instrucción *ARC* como la *ARC_R* pueden hacer mención a un canal específico en una ventana (window) señalada mediante un número.

ARC_R #6,50,60, **PI**/2

El siguiente programa ejemplo ilustra el uso de la sentencia ARC para el dibujo de óvalos de colores variados en la pantalla.

Ejemplo:

```
100  REMark    Prueba de la sentencia ARC
110  PAPER 0
120  CLS : CLS #0
130  REPeat ovalos
140       INK 6
150       a = RND (3 TO 145)
160       b = RND (1 TO 145)
170       c = RND (1 TO 45)
180       d = RND (1 TO 130)
190       AT 19,5
200       FOR i = 1 TO 2 STEP .15
210            INK RND (1 TO 7)
220            ARC a,b  TO  c,d + i,1
230            ARC c, d + i TO a,b,1
240       END FOR i
250       IF RND > .84  THEN CLS
260  END REPeat ovalos
```

9.13. LA INSTRUCCION FILL

Mediante esta instrucción puede colorearse interiormente una determinada figura cóncava con un color previamente definido.

El formato de la instrucción es:

FILL { 1 / 0 }

con 1 se activa el coloreado y con 0 se desactiva.

Ejemplo:

El grupo de instrucciones:

```
10  INK 4
20  FILL 1
30  CIRCLE 50, 60, 40
```
dibujara un circulo coloreado interiormente de color verde.

9.14. LA INSTRUCCION SCROLL

La sentencia SCROLL desplaza verticalmente la totalidad o cierta sección de la pantalla conectada al QL.

El formato de la instrucción es:

SCROLL [*canal,*] *expresión* [*,parte*]

donde *expresión:* debe ser una expresión numérica (o una constante o una variable del mismo tipo) y representa el nro. del pixels que se verán afectados por la instrucción.

parte: según las siguientes codificaciones:

0: la pantalla entera
1: arriba excluyendo la línea del cursor.
2: abajo excluyendo la línea del cursor.

Ejemplos:

```
10  SCROLL 20
```
desplaza hacia arriba 20 pixels

```
20  SCROLL – 50
```
desplaza hacia abajo 50 pixels

```
40  SCROLL -10,2
```
desplaza la parte baja de los diez pixels inferiores.

9.15. LA INSTRUCCION PAN

La instrucción PAN desplaza la pantalla *horizontalmente* un número de pixels determinado por la propia instrucción.

El formato de la instrucción es:

PAN [*canal,*] *distancia* [*,parte*]

donde *distancia:* representa el número de pixels que se verán afectados por la instrucción. Este parámetro es siempre obligatorio.

parte: representa qué sección de la pantalla se verá afectada por la instrucción según el siguiente código:

0: toda la pantalla
3: sólo la línea del cursor
4: a la derecha del cursor

Ejemplos:

```
20  PAN 50
```
desplaza la pantalla 50 pixels a la izquierda.

```
30  PAN -60
```
desplaza la pantalla 60 pixels a la derecha.

```
40  PAN 40,3
```
desplaza la línea del cursor 40 pixels a la izquierda.

9.16. LA INSTRUCCION CURSOR

La sentencia CURSOR permite posicionar al cursor en el lugar de la pantalla o de la ventana asociada deseado.

Esta sentencia hace referencia al sistema de ejes de coordenadas de los pixels; esto es, la posición del origen se encuentra en la esquina superior izquierda de la pantalla. Dependiendo del tamaño de letra utilizado en el instante considerado, así poseerá diferente tamaño dicho cursor.

El formato de la instrucción es:

CURSOR [*canal*,] *x*, *y* [,*x'*, *y'*]

donde *x*, *y*: son las coordenadas de la posición del cursor. Estos dos parámetros son obligatorios.
x', *y'*: son las coordenadas relativas al punto origen cuando se utiliza el sistema gráfico de coordenadas. Se trata de parámetros opcionales.

Ejemplos:

CURSOR 30,40

CURSOR 40,40,20,20
respecto de un hipotético origen de coordenadas
en el punto (40,40).

El siguiente programa ilustra el uso de la instrucción CURSOR. Se recomienda al lector que lo pruebe.

```
100  REMark  Prueba CURSOR
110  MODE 8
120  FOR bucle = 0 TO 400 STEP 2
130         CURSOR bucle, bucle/2
140         PRINT "Diagonal"
150  END FOR bucle
```

9.17. LA INSTRUCCION FLASH

La sentencia FLASH sólo puede ser utilizada en el modo 8 exclusivamente y producirá el efecto de hacer destellante lo que se visualice a continuación.

El formato de la instrucción es:

FLASH $\left\{ \begin{matrix} 1 \\ 0 \end{matrix} \right\}$

donde *1:* activa el parpadeo y
0: desactiva el parpadeo

Ejemplos:

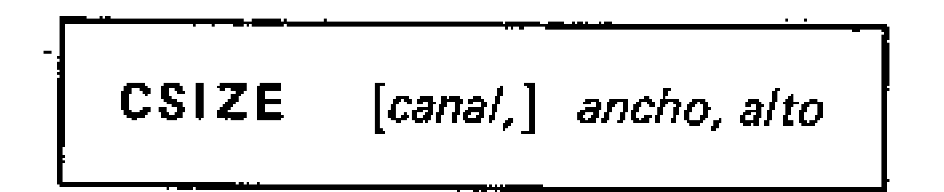
50 **FLASH** 1
activa el switch de parpadeo

50 **FLASH** 0
desactiva el parpadeo volviendo a la situación normal.

9.18. LA INSTRUCCION CSIZE

Mediante esta instrucción puede cambiarse el tamaño de los caracteres dentro de la visualización de la pantalla.

El formato de la instrucción es:



CSIZE [*canal*,] *ancho*, *alto*

donde *ancho:* puede poseer los valores 0, 1, 2 ó 3 de menor a mayor ancho.
alto: puede tomar los valores 0 ó 1 de menor a mayor altura.

Los tamaños por defecto en los dos modos de operación son:

MODE 4 .... CSIZE 0,0
(25 líneas de 84 caracteres)

MODE 8 .... CSIZE 1,0
(25 líneas de 42 caracteres)

Ejemplos:

20 **CSIZE** 3,1
todas las salidas posteriores seran visualizadas con el mayor tamaño de letra permitido.

20 **CSIZE** 0,0
lo mismo pero con el tamaño menor posible.

El cuadro de la figura 9-10 muestra el número de pixels utilizados para cada *ancho* y *alto* especificado.

ANCHO	TAMAÑO	ALTO	TAMAÑO
0	6 pixels	0	10 pixels
1	8 pixels	1	20 pixels
2	12 pixels		
3	16 pixels		

Fig. 9.10.– Tabla de tamaños de caracteres con la sentencia CSIZE.

9.19. LA INSTRUCCION STRIP

Mediante esta instrucción puede proporcionarse un fondo especial para hacer que los caracteres resalten más, mediante el uso de códigos de color ya conocidos.

El formato de la instrucción es:

STRIP [*canal,*] *color*

Ejemplos:

30 **STRIP** 7
proporciona un fondo de color blanco

40 **STRIP** 2,4,2
proporciona un fondo rojo/verde vertical

En esta instrucción se permite cualquier combinación de colores para su uso.

9.20. LA INSTRUCCION OVER

Habitualmente la impresión de los caracteres de determinado texto se visualizan utilizando el color de fondo actual en curso. Puede, no

obstante, ser alterada esta situación usando la instrucción STRIP anterior de la forma:

20 **OVER** 1
visualizara el texto en fondo transparente
seleccionado previamente

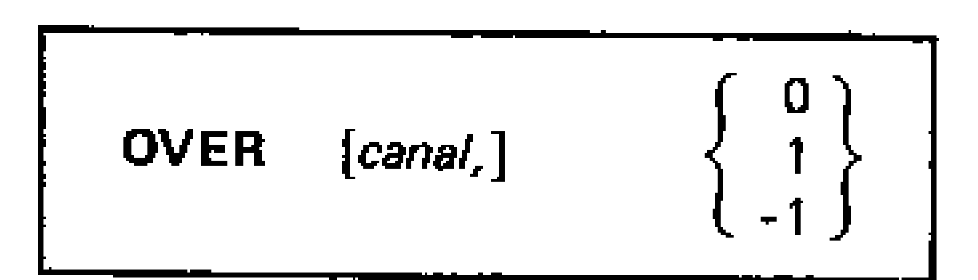
20 **OVER** – 1

visualizara en el fondo actual de la pantalla.

Para volver a la impresión normal deberá usarse:

30 **OVER** 0

Así pues, el formato general de esta sentencia es:



9.21. LA INSTRUCCION UNDER

Mediante la activación de esta instrucción se visualizarán los caracteres subrayados.

UNDER 1
subraya todos los caracteres que aparezcan
visualizados en la pantalla, en el color utilizado
hasta la aparición de

UNDER 0
que desactiva el subrayado, volviendo a la situación normal.

El formato general de la instrucción es:



9.22. LAS VENTANAS: LA INSTRUCCION WINDOW

Mediante esta sentencia se pueden crear o modificar ventanas de trabajo creadas en la pantalla o en el canal que se mencione.

El formato de la sentencia es:

```
WINDOW [canal,] ancho,alto,x,y
```

donde *ancho:* representa el número de pixels que tendrá de ancho la ventana.
alto: representa el número de pixels que tendrá de alto la ventana.
x,y: representan las coordenadas del origen de la ventana según el criterio del sistema de pixels de coordenadas.

Estos tres parámetros pueden ser o bien expresiones-numéricas, constantes o variables de este tipo.

Ejemplo:

```
30   WINDOW 20,30,15,15
creara una ventana de 20 pixels de ancho por
30 pixels de alto en el punto de coordenadas
(15,15)
```

9.23. ORGANIZACION DE LA PANTALLA

Como hemos dicho, la pantalla del QL se organiza en ventanas (windows) con las que se puede trabajar independientemente.

En el momento de conectar el QL y según se actúe sobre un monitor o un receptor normal de TV se obtendrán las dos posibles apariencias mostradas en la figura 9-11.

Las ventanas anteriores podrán ser identificadas como #0, #1 y #2 y estos guarismos podrán ser utilizados en cualquiera de los comandos vistos con anterioridad.

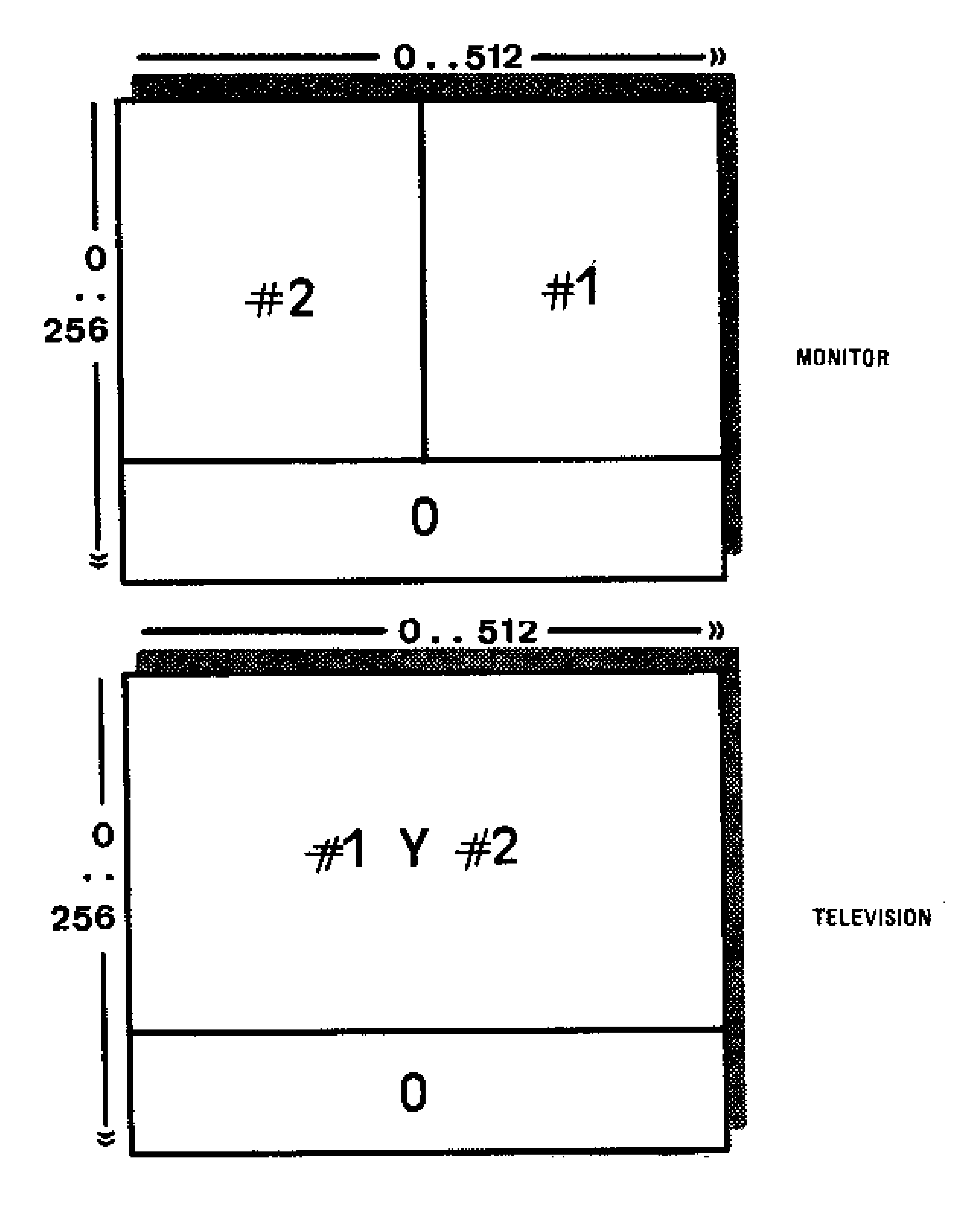


Fig. 9.11.— Organización de la pantalla.

Así:

CLS borrará la ventana # 1 (ventana por defecto)

pero

CLS 2 borrará la ventana #2

Palabra-clave	Efecto	Por defecto
AT	Posición del carácter	#1
BLOCK	Dibuja un rectángulo	#1
BORDER	Dibuja el borde	#1
CLS	Limpia la pantalla	#1
CSIZE	Tamaño de los caracteres	#1
CURSOR	Posición del cursor	#1
FLASH	Produce/cancela el parpadeo	#1
INK	Color de uso	#1
OVER	Efecto de impresión y gráficos	#1
PAN	Desplazamiento horizontal	#1
PAPER	Color de fondo	#1
RECOL	Cambia el color	#1
SCROLL	Desplazamiento vertical	#1
STRIP	Fondo para impresión	#1
UNDER	Subrayado	#1
WINDOW	Cambia la ventana actual	#1
LIST	Lista el programa	#2
DIR	Lista el directorio	#1
PRINT	Imprime caracteres	#1
INPUT	Entrada por teclado	#1

Fig. 9.12.— Instrucciones con número de canal y canal por defecto.

Lo mismo cabría decir de la instrucción PAPER.

PAPER #2,4

obtendrá el verde (código 4) de color de fondo sobre la ventana #2.

A los números #0, #1 y #2 se les denomina *números de canal* o simplemente *canales.*

La tabla de la figura 9-12 muestra una lista de las instrucciones que pueden utilizar un número de canal y el canal por defecto asumido cuando no se menciona ninguno.

9.24. LA GEOMETRIA DE LA TORTUGA

Utilizando las más actuales técnicas de diseño gráfico, el SuperBASIC del QL posee un sistema que se ha dado en llamar *geometría de la tortuga.* Este nombre proviene de los gráficos que son dibujados siguiendo el movimiento de una torgtuga imaginaria que se pasea por la pantalla. Así pues, todas las instrucciones de movimiento son *relativas* a la posición de la tortuga en ese momento.

La geometría de la tortuga tiene su origen en un lenguaje de programación llamado LOGO aún cuando su uso con SuperBASIC es mucho más simple.

La *tortuga* puede entonces moverse (hacia adelante o hacia atrás) produciendo (o no) un *rastro* dibujado de su movimiento. También puede girar su orientación a la derecha o a la izquierda un número especificado de grados.

Estudiaremos seguidamente las instrucciones para realizar dibujos utilizando este sistema.

9.25. LA INSTRUCCION MOVE

La instrucción MOVE hará que se desplace la tortuga por la pantalla o por el canal especificado, tantas posiciones como se indique en la propia instrucción.

El formato de la sentencia es:

MOVE [*canal,*] *distancia*

donde *distancia:* es el número de pasos que recorrerá la tortuga.

Mediante la escala elegida, así se moverá más o menos la tortuga.

Si se especifica una *distancia negativa* (precedida por un signo menos), la tortuga retrocederá; si la distancia es positiva, la tortuga avanzará.

Ejemplos:

MOVE #2,30
mueve la tortuga 30 unidades hacia adelante
y dentro del canal o ventana #2

MOVE –20
mueve la tortuga 20 unidades hacia atras.

9.26. LA INSTRUCCION PENUP

Se refiere al rastro que dibuja el recorrido de la tortuga en la pantalla o en el canal seleccionado.

El formato de la instrucción es:

```
PENUP [canal]
```

La instrucción PENUP hace "subir" el lapiz de dibujo del recorrido de la tortuga y por tanto, hasta que no vuelva a "bajarse" con la instrucción PENDOWN no se dibujará nada en la pantalla o en el canal elegido.

9.27. LA INSTRUCCION PENDOWN

Cuando se escribe esta instrucción, el lápiz "bajará" al papel que poseerá el color previamente definido y estará en condiciones de dibujar el recorrido que haga la tortuga (con el color de tinta previamente definido) hasta que se ejecute una instrucción PENUP.

El formato de la instrucción es:

```
PENDOWN [canal]
```

Como siempre, tanto en la instrucción PENUP como en la PENDOWN si no se especifica el canal, estas sentencias se referirán a la ventana por defecto utilizada.

9.28. LAS INSTRUCCIONES TURN Y TURNTO

Estas sentencias permiten que la tortuga gire sobre su eje un número determinado de grados, de forma que su posterior recorrido sea divergente con el anterior efectuado.

El formato de la instrucción TURN es:

```
TURN [canal,] ángulo
```


y girará el número de grados que se especifiquen en *ángulo*. Cuando se especifica un *ángulo negativo* este giro se hará según el movimiento de las agujas del reloj y *positivo* en caso contrario.

El formato de la sentencia TURNTO es:

```
TURNTO [canal,] ángulo
```

donde *ángulo* representará aquí el valor *absoluto* con el que quedará la tortuga después de su ejecución.

Ejemplos:

```
30 TURN 90
```

girará la tortuga 90 grados desde su posicion actual en sentido inverso a las manecillas del reloj.

```
40 TURNTO 0
```

la tortuga girará —si es necesario— hasta colocarse a cero grados respecto de la pantalla o del canal (ventana) elegido.

Inicialmente, la tortuga se encuentra a cero grados en la parte derecha de la pantalla o canal seleccionado.

Ejemplo:

Sean los dos procedimientos recursivos siguientes:

```
10  DEFine PROCedure inspiral (lado, angulo,incremento)
20      MOVE lado
30      TURN angulo
40      inspiral lado,lado + incremento, incremento
50  END DEFine
60  DEFine PROCedure expiral (lado, angulo, incremento)
70      MOVE lado
80      TURN ángulo
90      expiral lado + incremento, angulo, incremento
100 END DEFine
```

Mediante el uso de estos procedimientos recursivos (obsérvese que se llaman a sí mismos en las líneas 40 y 90) pueden realizarse curiosos dibujos utilizando la geometría de la tortuga.

Complétese el programa con las sentencias:

```
110 CLS
120 PENDOWN
130 POINT 50,50
```

y la línea siguiente puede ser alguna de las siguientes a modo de ejemplo:

```
140 inspiral 10,1,10
140 inspiral 10,1,20
140 inspiral 5,5,5
140 inspiral 5,5,20
140 inspiral 5,10,20
etc.
o bien:
140 expiral 1,60,1
140 expiral 1,100,1
140 expiral 1,117,1
140 expiral 1,120,1
140 expiral 1,150,1
140 expiral 1,179,1
```

Estas sucesivas pruebas no poseen final, por lo que el usuario deberá abortar la ejecución apretando las teclas CTRL y la barra de espaciado simultáneamente.

Seguidamente se muestra un programa ejemplo del uso de la geometría de la tortuga en la construcción de los más variados gráficos con la adición especial de sonido.

Ejemplo:

```
100 REMark   Graficos de la tortuga
110 PAPER 0
120 CLS : CLS #0
130 PENDOWN
140 REPeat bucle
150       IF RND > .68 THEN CLS
160       n = RND
170       POINT 80,50
```



```
180       INK RND (1 TO 7)
190       tor = RND (3 TO 70)
200       e = RND (25 TO 340)
210           FOR K = 1 TO tor
220               MOVE K/.7
230               TURN e
240               BEEP 3200, K/n
250           END FOR K
260       IF INKEY$ <> "" THEN STOP
270 END REPeat bucle
```

Este programa se detendrá apretando una tecla cualquiera.

10

Los ficheros

10.1. INTRODUCCION

Dentro de los microdrives del Sinclair QL pueden almacenarse tanto programas como ficheros de datos.

Como ya sabemos, un fichero puede ser definido como un conjunto de registros que contienen informaciones inherentes a un cierto tipo de unidad (p. ej. los datos referentes a una persona, automóvil, ordenador, etc.) con lo que se pueden tener diversos ficheros o archivos de personal, tráfico, informática, etc., respectivamente.

Existen fundamentalmente dos tipos de ficheros de datos según las organizaciones de los registros que los componen.

FICHEROS SECUENCIALES donde los campos que componen cada uno de sus registros son leídos normalmente en la misma secuencia en la que están dispuestos en el soporte magnético: discos, microdrives, etc., y comenzando por el primero de ellos. De esta manera, para localizar el registro en el fichero será necesario haber leído con anterioridad las anteriores en secuencia física.

FICHEROS DIRECTOS que tienen la posibilidad de acceder al registro deseado sin necesidad de leer ningún otro anterior o posterior. Este acceso es, por consiguiente, más rápido.

En los siguientes apartados de este capítulo estudiaremos con detalle todas las instrucciones y características especiales para la declaración y manejo de tales ficheros de datos.

10.2 DENOMINACION DE LOS FICHEROS

Para hacer referencia a un fichero es necesario suministrar dos informaciones:

dispositivo_nombrefichero

donde *dispositivo:* puede ser *mdv1* o *mdv2* según donde se halle situado el microdrive a utilizar.
nombrefichero: es el nombre propiamente dicho del fichero de datos y se definirá con este nombre dentro del directorio del microdrive utilizado.

Ejemplos:

mdv1_agenda
se menciona el fichero *agenda* que debe estar registrado en el microdrive *mdv1*

mdv2_micros
lo mismo con el fichero *micros* registrado en el microdrive *mdv2.*

10.3. APERTURA DE FICHEROS: La instrucción OPEN_NEW

Antes de poder utilizar un fichero, es necesario "abrirlo", es decir, dejarlo disponible para su uso, tanto si se trata de un fichero de nueva creación como uno que ya existiera.

Para crear un fichero de datos en SuperBASIC debe utilizarse la instrucción OPEN_NEW que posee el siguiente formato:

OPEN_NEW *canal,nombre*

donde *canal:* será el identificativo con el que será referenciado a partir de ese momento el fichero en cuestión y su estructura sintáctica es:

$$\# \left\{ \begin{matrix} 1 \\ 2 \\ \cdot \\ \cdot \\ \cdot \\ 15 \end{matrix} \right\}$$

es decir, pueden utilizarse simultáneamente en un programa hasta 15 ficheros distintos.

nombre: de la forma:

$$\left\{ \begin{matrix} \text{mdv1_} \\ \text{mdv2_} \end{matrix} \right\} \quad \text{nombrefichero}$$

como se ha visto en el apartado anterior.

10.4. CIERRE DE FICHEROS: La instrucción CLOSE

Cuando se ha acabado de utilizar un fichero, también es necesario "cerrarlo". Para ello, se utiliza la instrucción CLOSE que posee el formato:

CLOSE *canal*

Ejemplo:

Crear un fichero que contenga los primeros 50 números naturales.

```
10  REMark ———————————————————
20  REMark        Creacion de fichero
30  REMark ———————————————————
40  OPEN_NEW #5,mdv1_numeros
50  FOR n = 1 TO 50
60        PRINT #5,n
70  END FOR n
80  CLOSE #5
```


Observemos algunos aspectos del programa anterior. En la línea 40 se ha abierto un fichero de datos llamado *mdv1_números* (colocado en un microdrive situado en el slot del mdv1) y que se asocia con el canal # 5.

La instrucción *60 PRINT # 5,n* procede a grabar en el canal especificado el contenido de la variable *n*, esto es, en el fichero abierto con anterioridad.

Por último, la línea 80 cierra el fichero manteniéndolo en el microdrive mencionado para posteriores usos.

De esta manera, cuando se realice una visualización del directorio del microdrive seleccionado, ya aparecerá el fichero "números" como parte integrante del mismo microdrive.

10.5. LECTURA DE FICHEROS: LA INSTRUCCION OPEN_IN

Cuando se desea leer exclusivamente un fichero, deberá utilizarse la instrucción OPEN_IN de la forma similar:

OPEN_IN *canal,nombre*

Ejemplo:

Leer visualizando en pantalla cada uno de los registros del fichero anerior.

```
10  REMark ————————————————
20  REMark   Visualizacion de fichero
30  REMark ————————————————
40  OPEN_IN #3, mdv1_numeros
50  FOR i = 1 TO 50
60        INPUT #3,n
70        PRINT n
80  END FOR i
90  CLOSE #3
```

Obsérvese en el programa anterior que el mismo fichero ha sido montado esta vez en el microdrive 2 y que se le ha asignado el canal # 3 (línea 40).

La línea 60 lee los registros del fichero anterior mediante una instrucción INPUT asociada al canal # 3 anterior.

En los ejemplos de programas anteriores hemos utilizado los canales # 5 y # 3 para denotar un fichero de datos ubicado físicamente en un microdrive, no obstante, existen ciertos dispositivos que tienen permanentemente asignados ciertos canales. El cuadro de la figura 10.1 muestra dichos dispositivos.

CANAL	USO
# 0	OUTPUT - comandos de ventana INPUT - teclado
# 1	OUTPUT - ventana de visualizacion
# 2	LIST - salida listada

Fig. 10.1.— Canales asociados permanentemente.

Como puede observarse en el cuadro anterior, puede tratarse a la pantalla del QL como un fichero más con un número de canal asociado y unas operaciones posibles permitidas en él. El apartado siguiente nos ayudará a entrar en estos conceptos de pantalla.

Después de haber creado un fichero nuevo en un microdrive (p. ej. con OPEN_NEW # 5, mdv1_fichero) al ejecutar el comando DIR (1).

DIR *mdv1_*

aparecerá *fichero* como un archivo de datos contenido en el microdrive mencionado.

(1) Nota: Tanto el comando DIR como el COPY serán estudiados con detalle en el Apéndice A de este texto.

Para visualizar en pantalla el contenido de un fichero creado en un microdrive puede utilizarse el comando (1).

COPY *mdv1_fichero* **TO SCR_**

sin necesidad de crear un programa para la visualización de cada uno de sus registros.

10.6. REGISTROS CON VARIOS CAMPOS

Análogamente pueden construirse ficheros cuyos registros posean varios campos.

Ejemplo:

Escribiremos un programa que cree un fichero en microdrive que sea una agenda telefónica donde cada registro contenga el nombre de la persona y su teléfono.

```
100 REMark------------------
110 REMark    Creacion de un fichero
120 REMark------------------
130 OPEN_NEW #7,mdv1_agenda
140 FOR i = 1 TO 10
150       INPUT nombre$,telefono
160       PRINT #7,nombre$,telefono
170 END FOR i
180 CLOSE #7
```

Con este programa puede construirse un fichero de 10 registros, donde cada uno de ellos poseerá dos campos: el nombre y el teléfono.

Cuando se ejecuta el programa anterior (RUN) se tecleará el nombre seguido de ENTER y el teléfono seguido de ENTER y todo ello diez veces.

Es importante señalar que los datos son almacenados en memoria hasta que el sistema está listo para transferirlos al microdrive especificado. Este acceso, por tanto, sólo se realiza una vez.

Ejemplo:

La siguiente versión del programa anterior permite introducir un *número variable* de registros en el fichero.

```
100  OPEN_NEW #7,mdv1_agenda
110  REPeat escribir
120        INPUT nombre$,telefono
130        IF nombre$ = "99" THEN EXIT escribir
140        PRINT #7,nombre$,telefono
150  END REPeat escribir
160  CLOSE #7
```

Así, cuando se teclee "99" en el campo *nombre$* la instrucción IF detectará el final del fichero y procederá a cerrarlo convenientemente.

10.7. FINAL DE FICHERO: LA FUNCION EOF

La manera de leer o utilizar un fichero del que a priori se desconoce el número de registros que tiene es similar. Se utilizará en este caso la función incorporada EOF cuyo formato es:

EOF *(canal)*

que dará como resultado el valor lógico *true* si se ha llegado al final del fichero y *false* en caso contrario.

Ejemplo:

El siguiente programa leerá visualizando en pantalla el fichero creado en el programa anterior, desconociendo el número de registros que posee.

```
100  REMark ------------------
110  REMark    Lectura de un fichero
120  REMark ------------------
130  OPEN_IN #6,mdv1_agenda
140  REPeat leer
150        INPUT #6,nombre$,telefono
160        IF EOF (#6) THEN EXIT leer
```



```
170        PRINT nombre$ ! telefono
180  END REPeat leer
190  CLOSE #6
```

Como es obvio, puede utilizarse más de una variable de string dentro de una instrucción INPUT o PRINT asociada a un fichero. No obstante, también puede leerse un fichero depositando el contenido de *todos* los campos del registro en una única variable de string y luego actuar sobre ella en consecuencia.

Ejemplo:

```
100  OPEN_IN  #5, mdv1_agenda
110  REPeat leer
120        INPUT #5, registro$
130        IF EOF (#5) THEN EXIT leer
140        PRINT registro$
150  END REPeat leer
160  CLOSE #5
```

En este programa, después de cada lectura (INPUT # 5) la variable *registro$* contendrá todos los campos que confirman el registro de datos, esto es, los correspondientes a *nombre$* y *teléfono*. Puede, por consiguiente, utilizarse la variable *registro$* de forma adecuada para acceder a cada campo específico.

10.8. CLASIFICACION DE FICHEROS (SORTING)

Para clasificar un fichero previamente creado en un microdrive, por uno o varios de los campos que componen cada registro, debe leerse previamente, cargando su contenido en memoria en forma de matriz de forma que se clasifique dicha matriz y se obtenga como salida un nuevo fichero que estará ya clasificado.

Siempre está recomendado que se haga una copia de seguridad del fichero a clasificar antes de procesarlo según los pasos anteriores.

Ejemplo:

El siguiente procedimiento sirve para ilustrar la clasificación de una tabla en memoria.

```
200 DEFine PROCedure clasificar (entrada,salida)
210    numero = entrada (0)
220    FOR i = 2 TO numero
230        p = i
240        entrada(0) = entrada(p)
250        REPeat comparar
260          IF entrada (0) > = entrada(p-1) THEN EXIT
270          entrada(p) = entrada (p-1)
280          p = p -1
290        END REPeat comparar
300        entrada(p) = entrada(0)
310    END FOR i
320    FOR k = 1 TO 8: salida(k) = entrada(k)
330 END DEFine
```

Este procedimiento recibe una matriz *(entrada)* que contiene los datos a clasificar. El elemento cero de dicha matriz contiene el número de elementos a clasificar. Como puede observarse, no se hace referencia a que los elementos de la matriz sean de tipo numérico o string. El principio de *conversión (coerción)* vistos en el capítulo 2 hará, si es necesario, las oportunas conversiones.

Obsérvese también en el procedimiento anterior que se ha presupuesto que el número de elementos a clasificar es de 8 (línea 320).

El procedimiento anterior puede probarse con las siguientes líneas de programa que presentan una matriz de strings con nombres propios a clasificar.

```
100 REMark -------------------
110 REMark      Prueba de clasificacion
120 REMark -------------------
130 DIMension nombre$ (7,7),regreso$(7,7)
140 nombre$(0) = 8
150 FOR k = 1 TO 8: READ nombre$(k)
160 clasificar nombre$,regreso$
170 PRINT regreso$ !
180 DATA "isabel","carlos","cordero","galan"
190 DATA "blas","felix","vicky","paco"
```

En el programa siguiente se muestra un ejemplo completo de clasificación de un fichero.

Obsérvese que se han utilizado tres procedimientos:

- *lectura:* lee un fichero que puede contener un número indeterminado de registros (limitado a 50 por las dimensiones de las matrices escritas en el programa) y lo introduce en una matriz en memoria.

- *clasificar:* Clasifica ascendentemente la matriz anterior con los campos del registro.

- *escritura:* Graba en microdrive un nuevo fichero clasificado partiendo del original.

```
100 REMark -------------------
110 REMark      Lectura, clasificacion y
120 REMark      escritura de un fichero
130 REMark -------------------
140 DIMension dato$(50,7), resultado$(50,7)
150 lectura
160 clasificar dato$,resultado$
170 escritura
180 STOP
190 REMark -------------------
200 REMark         Procedimientos
210 REMark -------------------
220 DEFine PROCedure lectura
230    OPEN_IN #5,mdv1_fichero
240    i = 1
250    REPeat leer
260      INPUT #5, dato$(i)
270      IF EOF (#5) THEN EXIT leer
280      i = i + 1
290    END REPeat leer
300    dato$(0) = i
310    CLOSE #5
320 END DEFine
330 REMark -------------------
340 DEFine PROCedure clasificar (entrada,salida)
350    numero = entrada (0)
360    FOR i = 2 TO numero
370        p = i
380        entrada (0) = entrada (p)
390        REPeat comparar
```

```
400         IF entrada (0) > = entrada (p-1) THEN EXIT comparar
410         entrada (p) = entrada (p-1)
420         p = p - 1
430       END REPeat comparar
440       entrada(p) = entrada (0)
450     END FOR i
460     FOR k = 1 TO numero: salida(k) = entrada(k)
470 END DEFine
480 REMark ----------------------
490 DEFine PROCedure escritura
500     OPEN_NEW #6,mdv1_clasificado
510     i = 2
520     REPeat escribir
530       PRINT #6,resultado$(i)
540       IF i = dato$ (0) THEN EXIT escribir
550       i = i + 1
560     END REPeat escribir
570     CLOSE #6
580 END DEFine
```

El usuario podrá modificar levemente los procedimientos anteriores para adecuarlos a su caso particular.

10.9. TIPOS DE FICHEROS

Existen en el QL varios tipos de ficheros:

DATA: Se trata de programas SuperBASIC, ficheros de texto, etc.
Se crean utilizando los comandos PRINT y SAVE y se accede a ellos utilizando INPUT, INKEY$, LOAD, etc.

EXEC: Se trata de programas ejecutables.
Se salvan utilizando el comando SEXEC y se cargan usando EXEC, EXEC_W, etc.

CODE: Se trata de porciones de datos en memoria, imágenes de pantalla, etc.
Se salvan utilizando SBYTES y se cargan usando LBYTES.

10.10. CARGA DE UN FICHERO EN POSICIONES ESPECIFICAS DE MEMORIA. EL COMANDO LBYTES

El comando LBYTES permite cargar un fichero de datos desde un microdrive en una dirección de comienzo de memoria específica.

El formato de este comando es:

LBYTES *dispositivo,dirección*

donde *dispositivo:* es el nombre del microdrive utilizado para la carga del fichero de datos.
dirección: puede ser una constante numérica o una variable que representa el valor de la dirección de comienzo de memoria, donde se cargará dicho fichero de datos.

Ejemplos:

LBYTES mdv1_pant,131072
que cargará una imagen de pantalla

LBYTES mdv2_prg,dirección
que cargará el programa *prg* en la dirección especificada

10.11. DESCARGA DE SEGMENTOS DE MEMORIA: EL COMANDO SBYTES

El comando SBYTES permite descargar en un dispositivo cualquiera del QL (los microdrives, por ejemplo) posiciones específicas de memoria.

El formato del comando es:

SBYTES *dispositivo,dirección,longitud*

donde *dispositivo:* representa el nombre del dispositivo utilizado para la salvaguarda del segmento de memoria especificado.

dirección: debe ser una expresión-numérica y que representa la dirección de comienzo de la zona de memoria a salvar.
longitud: debe ser también una expresión-numérica y que representa la longitud del segmento de memoria que debe ser salvada.

Ejemplos:

SBYTES mdv1_pantalla,131072,32768
salva la pantalla de direccion 131072 y longitud
32668 bytes en el dispositivo especificado.

SBYTES mdv2_programa,50000,10000
salva el programa de direccion de memoria 50000
y longitud 10000 bytes en el dispositivo mencionado.

SBYTES SER1,0,32768
salva la memoria hasta la direccion 32768 en el
canal en serie SER1 mencionado.

10.12. FICHEROS DE PANTALLA Y DE TECLADO

Ya hemos visto en el capítulo 9 la definición de ventana dentro del contexto de la pantalla.

Puede crearse una ventana de cualquier tamaño en cualquier lugar de la pantalla. El nombre asociado con tal dispositivo es

SCR_

El formato general de un dispositivo SCR_ es:

SCR_ [*ancho x alto*] [*a X x Y*]

donde *ancho y alto:* corresponden a las medidas de la ventana (window) seleccionada.
x, y: representan las coordenadas de origen de la ventana (window) seleccionada.

Observe el lector el programa siguiente:

```
10  REMark ----------------------
20  REMark        Creacion de una ventana
30  REMark ----------------------
40  OPEN #4, SCR_ 360x150a80x40
50  PAPER #,4,4
60  CLS #4
70  CLOSE #4
```

En la línea 40 del programa anterior se ha abierto una ventana asignada al canal # 4 con un ancho de 360 pixels y una altura de 150, situado en unos ejes de coordenadas de 80 de abscisa y 40 de ordenadas.

En la línea 50 del programa se ha seleccionado el color de fondo para la ventana creada con anterioridad. En la línea 60 se procede a limpiarla de todo posible contenido y, por último, en la línea 70 se cierra (CLOSE) la ventana como si de un fichero de datos tradicional se tratara.

Para cambiar las características de una ventana previamente creada (OPEN), no es necesario volver a crearla. Para ello existe el comando WINDOW con una sintaxis similar a la OPEN y que permite modificar la anchura, altura y situación de la ventana.

Ejemplo:

Para la misma ventana creada con el programa anterior podría escribirse:

```
65  WINDOW #4,300,110,100,65
66  PAPER #4,2
67  CLS #4
```

que cambiará el valor de los atributos de la ventana anterior asociada al mismo canal #4.

Tal y como ya veíamos en el capítulo 9 pueden utilizarse las instrucciones PAPER, BORDER, BLOCK, etc., asociadas con un canal que será una ventana previamente creada.

Ejemplos:

```
20    BORDER #4,6
```

creará un borde alrededor de la ventana #4 de 6 unidades de ancho

```
20    BORDER #4,6,2
```

lo mismo, pero esta vez el borde es de color rojo

```
40    BLOCK #4,10,20,50,100,2
```

creará un rectángulo en la ventana #4 en las coordenadas (50,100) y que poseerá 10 unidades de ancho y 20 unidades de alto, siendo de color rojo.

También puede hablarse de ficheros de teclado (introducidos por el teclado del QL) mediante el nemotecnico CON_ que posee el formato general:

CON_ [ancho x alto] [aXxY] [_k]

donde *ancho y alto:* son las medidas representativas de la ventana (window) seleccionada.
X, Y: son las coordenadas de la ventana seleccionada.
k: tipo de teclado y longitud del buffer.

Ejemplos:

```
OPEN #4, CON_20x50a0x0_32
OPEN #6, CON_
```

Estos ficheros de teclado pueden utilizarse de manera análoga a los ficheros tradicionales con el uso de INPUT y CLOSE.

El siguiente programa ejemplo ilustra el uso de la creación, en este caso, de cuatro ventanas simultáneas en la pantalla donde aparecerá moviéndose el texto dado.

Ejemplo:

```
100  REMark   Prueba de creacion de ventanas
110  PAPER 1: CLS : CLS #0
```



```
120  a$ = "En un lugar de la Mancha de cuyo nombre"
130  a$ = a$ & a$
140  OPEN   # 5, CON_ 130x130a90x15
150  OPEN   # 6, CON_ 130x130a60x60
160  OPEN   # 7, CON_ 130x130a205x105
170  OPEN   # 8, CON_ 130x130a180x15
180  REPeat bucle
190       PAPER  # 5,7 : CLS #5
200       PRINT  # 5, a$ (1 TO RND (100 TO 418))
210       PAPER  # 6,6 : CLS #6
220       PRINT  # 6, a$ (1 TO RND (100 TO 418))
230       PAPER  # 7,3 : CLS #7
240       PRINT  # 7, a$ (1 TO RND (100 TO 418))
250       PAPER  # 8,2 : CLS #8
260       PRINT  # 8, a$ (1 TO RND (100 TO 418))
270       IF INKEY$ <> " "  THEN EXIT bucle
280  END REPeat bucle
290  CLOSE #5 : CLOSE #6
300  CLOSE #7 : CLOSE #8
```

11

Algunos conceptos complementarios

11.1. INTRODUCCION

En este capítulo se estudian algunos aspectos complementarios para la programación adecuada del Sinclair QL.

En primer lugar se detalla el uso del color y su construcción interna, aprendiendo cómo se forman los diferentes tonos e intensidades y su uso. Seguidamente se estudian las comunicaciones en serie vía el interfase RS-232C que posee el QL. Se trata también el reloj interno del sistema y se describe cómo utilizar y actualizar dicho reloj, escribiendo los diferentes comandos e instrucciones de uso.

Se relaciona también una lista completa y traducida de todos los diagnósticos de error que pueden producirse en tiempo de interpretación y/o de ejecución durante un programa.

Se estudia también la estructura de la memoria efectiva y potencial del QL, así como el uso y mantenimiento de los microdrives y las redes de comunicación. El siguiente apartado está dedicado íntegramente al estudio de diversas funciones y comandos de uso del sistema operativo QDOS y por último dedicamos una breve sección final al estudio de las facilidades sonoras del ordenador con las sentencias de uso contempladas en la sintaxis del SuperBASIC.

11.1. EL COLOR

En la tabla de la figura 9-3 ya estudiábamos cuántos y qué colores podían ser utilizados para el diseño gráfico en la pantalla conectada al QL. Veámos ahora algo más acerca de la composición del color en el ordenador.

Los colores en el QL pueden estar formados bien por un color *definido* o bien por una *mezcla* de un color definido sobre un cierto *granulado*. Así pues, podemos decir que una especificación completa de un color estaría formada por tres aspectos:

1. El color definido
2. El contraste
3. El granulado elegido

Cada uno de estos tres parámetros puede ser programado independientemente en el QL y ocupa un byte de memoria con la estructura que se muestra en la figura 11-1.

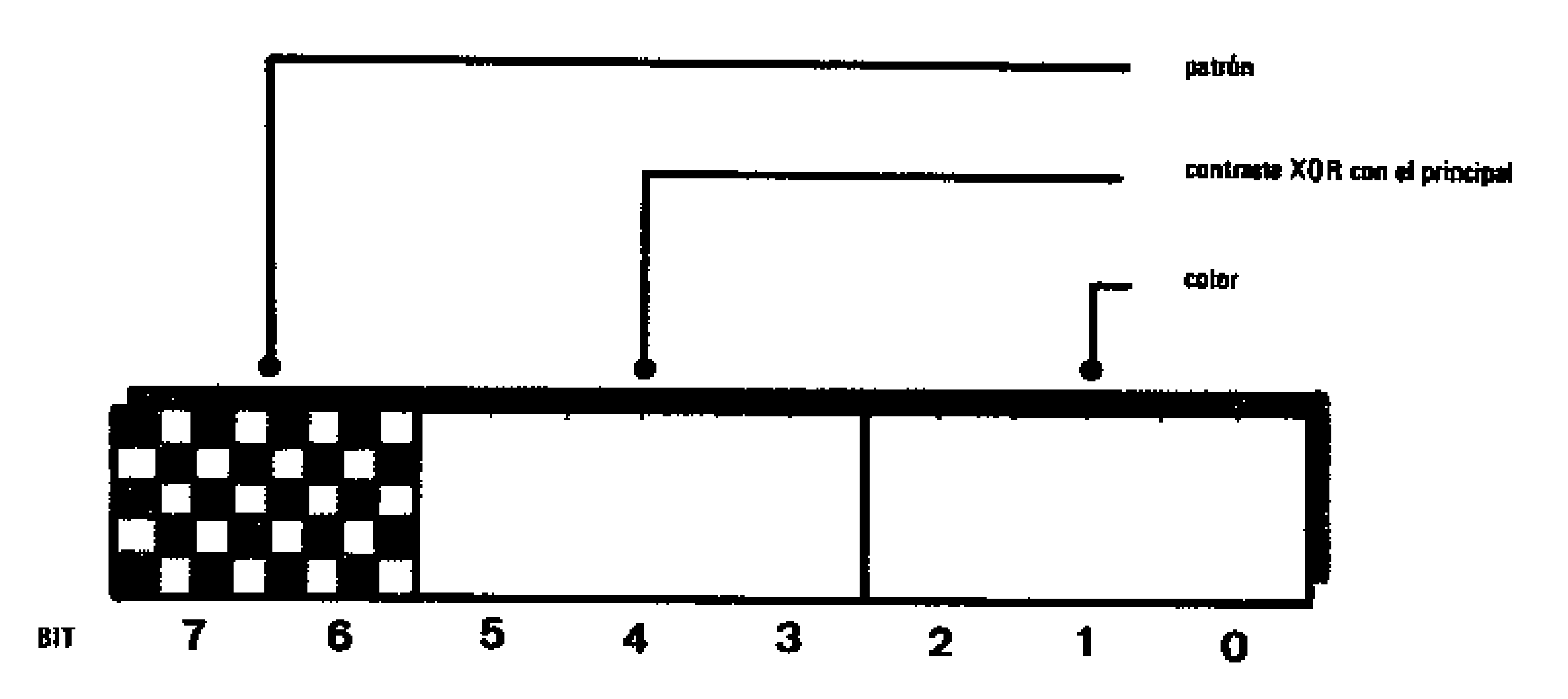


Fig. 11.1.— Estructura general del byte de color.

De esta forma, siempre es posible especificar un código de color utilizando tres parámetros:

color := definido , contraste , granulado

En la figura 9-4 se muestran todos los posibles tipos de granulado, siendo el número 3 el grano por defecto cuando no se especifica ninguno.

Ejemplos:

Estudie el lector mediante sucesivas experimentaciones la utilización de estos colores:

```
PAPER 255
PAPER 2,4
PAPER 0,2,0
```

El siguiente programa ejemplo muestra la gran cantidad de superficies de colores diferentes que pueden obtenerse con su QL. Si Vd. dispone de un monitor en color en vez de un televisor doméstico, podrá apreciar esto con mayor definición.

Ejemplo:

```
100 REMark    Prueba de colores
110 MODE 256
120 CSIZE 3,1
130 FOR fondo = 7 TO 0 STEP -1
140       FOR contraste = 0 TO 7
150             FOR moteado = 0  TO  3
160                   PAPER  fondo, contraste, moteado
170                   CLS : AT 2,5
180                   PRINT  "fondo" ! 'fondo \\
190                   PRINT  "contraste" ! contraste  \\
200                   PRINT  "moteado" ! moteado
210                   PAUSE 25
220             END FOR  moteado
230       END FOR contraste
240 END FOR  fondo
```

11.2. INTERFACE RS-232C

El Sinclair QL dispone, como dijimos en el capítulo introductorio, de dos puertas seriales (denominadas SER1 y SER2) que están configuradas de forma algo diferente. Fundamentalmente una de ellas está configurada para su conexión con un modem y la otra para una impresora con interface en serie.

La velocidad de transmisión entre el QL y los dispositivos periféricos conectados puede ajustarse con el comando BAUD que posee el formato:

BAUD *velocidad*

donde *velocidad:* puede ser bien una constante o una variable o incluso una expresión-numérica, y que tendrá que tomar uno cualquiera de los valores:

75
300
600
1200
2400
4800
9600
19200 (sólo en transmisión)

La transmisión por defecto es de 9600.

Si la velocidad de transmisión seleccionada no está soportada, se genera un error.

Ejemplos:

```
BAUD 4800
BAUD velocidad_impresión
```

La referenciación a un canal en serie puede hacerse según el formato:

SER *n* [*p*] [*h*] [*z*]

donde *n*: es el número de puerta y puede ser 1 ó 2.
p: es la verificación de la *paridad* para la detección de posibles errores de transmisión, según la codificación:

e – par
o – impar
m – marca
s – espacio

h: es la línea de oscilación del modem para adaptadores de línea oscilantes, según la codificación:

i – ignorado
h – oscilación

z: es el *protocolo* utilizado, según la codificación:

r – sin EOF
z – CTRL_Z es EOF
c – (CR) es separador de registros

La forma por defecto es:

SER1r

Ejemplos:

OPEN #6, SER1e
COPY mdv2_agenda **TO SER1c**

Como se ha visto en los ejemplos anteriores, deben utilizarse las instrucciones OPEN y CLOSE para abrir o cerrar respectivamente los canales en serie asignados.

11.3. ANCHURA DE CANALES: La instrucción WIDTH

Con esta sentencia se puede fijar el ancho de operación con el que trabajarán los distintos dispositivos periféricos sujetos a esta instrucción.

Pueden utilizarse, por ejemplo, para fijar la anchura del número de columnas a escribir en una impresora.

El formato de instrucción es:

WIDTH *[canal,] ancho*

donde *ancho:* representa la anchura de la línea y puede ser una expresión numérica, una constante o una variable del mismo tipo.

Ejemplos:

WIDTH 100
se fija la anchura del dispositivo en 100.

WIDTH #7,130
se fija en 130 la anchura del canal asociado al número #7.

11.4. EL RELOJ DEL SISTEMA

El Sinclair QL dispone de un reloj interno de tiempo real que se encuentra activo cuando el aparato está encendido.

El QL dispone de varias funciones e instrucciones para el acceso y modificación de este reloj.

11.4.1. Las instrucciones ADATE, DATE$, DATE y DAY$

La instrucción ADATE permite ajustar el reloj interno, adelantándolo o atrasándolo.

El formato de la instrucción es:

ADATE segundos

Ejemplos:

ADATE 7200
avanzara el reloj 2 horas.

ADATE –120
atrasara el reloj 2 minutos

En el formato anterior, *segundos* puede ser bien una constante numérica o bien una variable o expresión del mismo tipo.

Puede accederse indistintamente a la fecha y a la hora utilizando la instrucción DATE$ y DATE.

DATE$ es una función incorporada que devuelve la fecha y la hora contenida en el propio reloj del sistema operativo del QL.

El formato de utilización de la función DATE$ es simple:

DATE$

sin ningún tipo de parámetro.

El formato de los datos que devuelve DATE$ es:

aaaa mm dd hh:mm:ss

donde *aaaa*: representan los cuatro dígitos del año en curso.
mm: representa un apócope del nombre del mes (p. ej. JAN por Enero, FEB por Febrero, etc.). Todas estas contracciones provienen naturalmente del idioma inglés, aunque utilizando las instrucciones adecuadas para el manejo de cadenas, pueden ser convertidas con facilidad al castellano.
dd: representa el día del mes de que se trate.
hh: representa la hora dentro de un intervalo comprendido entre 0 y 23.
mm: representa los minutos dentro de un intervalo comprendido entre 0 y 59.
ss: lo mismo con los segundos.

Por otro lado, la función DATE devuelve también la fecha y la hora, pero de una forma *compacta* de manera que sea susceptible de ser almacenada como un número real y de posibilitar el almacenaje y la comparación, por ejemplo, entre varias fechas.

El formato de esta función es igualmente simple:

DATE

sin ningún tipo de parámetro.

La función DATE$ puede ir acompañada de un parámetro. Cuando esto sucede, este parámetro será tomado como una fecha escrita en forma compacta y la función DATE$ la convertirá a forma descompactada de cadena y como veíamos al principio.

Ejemplos:

10 PRINT DATE$

20 PRINT DATE$ (fecha)
convierte el valor de *fecha* a forma de cadena.

El QL dispone, además, de otra función de reloj que es capaz de devolver el día de la semana de la fecha.

El formato de esta función es:

DAY$ [(fecha)]

DAY$ es una función que devuelve el día de la semana de la fecha actual. Cuando la función DAY$ es acompañada de un parámetro, este parámetro será tomado como una fecha en forma compacta y serán extraídos y convertidos a cadena el día de la semana.

Ejemplos:

10 PRINT DAY$

20 PRINT DAY$ (fecha)
convierte el valor de *fecha* a forma de cadena e imprime el día de la semana.

11.4.2. La instrucción SDATE

Mediante esta instrucción puede fijarse la fecha y la hora en el reloj del sistema del QL.

El formato de la sentencia es:

SDATE *año,mes,día,horas,minutos,segundos*

donde cada uno de estos parámetros pueden ser, en el caso más general, una expresión numérica o bien una constante o una variable del mismo tipo.

Ejemplos:

SDATE 1986,10,2,3,1,5

11.5. LOS DIAGNOSTICOS DE ERROR

Los diagnósticos o mensajes de error son suministrados por el Super BASIC cuando no puede completar una operación cualquiera de las explícitamente mencionadas en un programa o comando.

El formato de los errores es:

AT LINE *xx texto*

donde *xx:* es el número de línea donde se ha producido el error.
texto: es la explicación del error encontrado.

La siguiente lista muestra los posibles errores que pueden presentarse durante el trabajo con un programa SuperBASIC, convenientemente traducidos para su interpretación sencilla.

(1) **NOT COMPLETE**
Una operación ha tenido que ser terminada prematuramente.

(2) **INVALID JOB**
Un error de retorno del QDOS mientras se estaba controlando un trabajo de Entrada/Salida o multitarea.

(3) **OUT OF MEMORY**
El QDOS y/o el SuperBASIC no tienen suficientemente memoria libre.

(4) **OUT OF RANGE**
Se ha intentado escribir fuera del ámbito de una ventana (window) o bien se ha intentado acceder a una matriz con un subíndice incorrecto.

(5) **BUFFER FULL**
Se ha llenado el buffer de Entrada/Salida antes de que se termine la operación.

(6) **CHANNEL NOT OPEN**
Se ha intentado leer, escribir o cerrar un canal que no está previamente abierto. Puede ocurrir también cuando se pretende abrir un canal erróneo.

(7) **NOT FOUND**
No puede encontrarse un fichero de sistema o un dispositivo con ese nombre. Puede tratarse también de que el SuperBASIC no pueda encontrar un identificador adecuado dentro del contexto de una instrucción, p. ej. en estructuras jerarquizadas.

(8) **ALREADY EXISTS**
El fichero mencionado en la instrucción ya existe catalogado con el mismo nombre.

(9) **IN USE**
Se ha pretendido utilizar un fichero o un dispositivo que se encuentra en uso exclusivo.

(10) **END OF FILE**
Se ha detectado un final de fichero de entrada.

(11) **DRIVE FULL**
El microdrive mencionado está lleno.

(12) **BAD NAME**
Existe un error de sintaxis en uno de los parámetros encontrados en la instrucción. En SuperBASIC significa que un determinado nombre ha sido usado fuera de su contexto.

(13) **XMIT ERROR**
Error de paridad en comunicación RS-232C

(14) **FORMAT FAILED**
Error durante una operación de formateo. El dispositivo puede estar estropeado.

(15) **BAD PARAMETER**
Se ha encontrado un error en una lista de parámetros (p. ej. de un procedimiento o de una función). Se ha intentado leer datos desde un dispositivo de salida.

(16) **BAD MEDIUM**
El dispositivo puede estar estropeado.

(17) ERROR IN EXPRESSION
Se ha encontrado un error durante la evaluación de una expresión.

(18) OVERFLOW
Se ha detectado un overflow o se ha intentado una división entre cero, la raíz cuadrada de un número negativo, etc.

(19) NO SE USA

(20) READ ONLY
Se ha intentado escribir un fichero que está catalogado como de sólo lectura.

(21) BAD LINE
Se ha detectado un error sintáctico en una instrucción SuperBASIC.

Cuando se detecta un error durante la ejecución de un programa, puede ser obviado saltando a la siguiente instrucción tecleando:

CONTINUE

Si el error puede subsanarse en tiempo de ejecución, sin cambiar el programa, entonces puede reintentarse la ejecución desde el punto de parada tecleando:

RETRY

Debe tenerse presente que un programa solamente podrá continuar si no se han añadido nuevas líneas al programa ni nuevas variables. Tampoco deben haberse modificado ninguna de las líneas del programa.

11.6. LA ESTRUCTURA DE LA MEMORIA

Como se verá en el Apéndice B donde se estudian las características más importantes de los microprocesadores que conforman el QL, el microprocesador Motorola 68008 con estructura interna de 32 bits es el que dirige toda la gestión de los recursos del sistema y está potencialmente diseñado para acceder a una memoria de 1 Megabyte, es decir, desde la posición 00000 a la FFFFF (en hexadecimal).

Estas direcciones son usadas por la máquina tal y como se muestra en la figura 11.2.

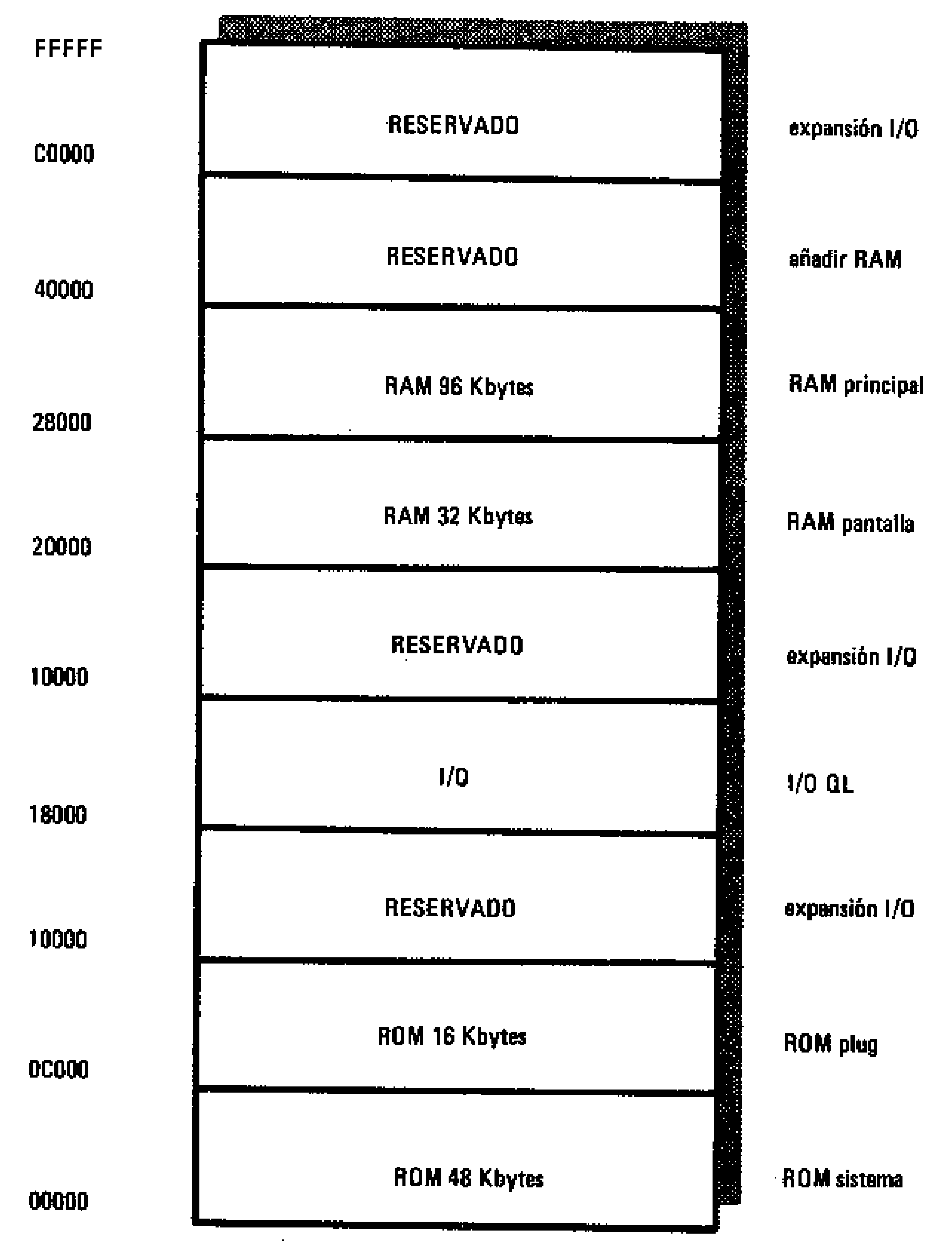


Fig. 11.2.– Estructura de la memoria del QL.

La pantalla (screen) del QL está organizada en palabras de 16 bits y como puede observarse en la figura anterior, comienza en la posición 20000.

Se recomienda al lector la lectura detallada del manual del usuario para un estudio más profundo de la estructura interna de la memoria del QL.

11.7. LOS MICRODRIVES

Los microdrives son los dispositivos de almacenamiento externo de programas y datos que posee de manera incorporada el QL. Aproximadamente la capacidad de almacenamiento de cada unidad de estas es de unos 100 Kbytes.

Antes de poder proceder a la introducción de datos o de programas en un microdrive nuevo es necesario inicializarlo, es decir, *formatearlo*, de forma que se divida su área de almacenamiento en *sectores* (como máximo 255) susceptibles de ser rellenos con las informaciones precisas.

Cada sector de los anteriores posee una capacidad de 512 bytes.

La manera de formatear un microdrive para su uso se estudia con detalle en el Apéndice A dedicado a los comandos SuperBASIC de uso.

La figura 11-3 muestra el aspecto que posee un microdrive standard.

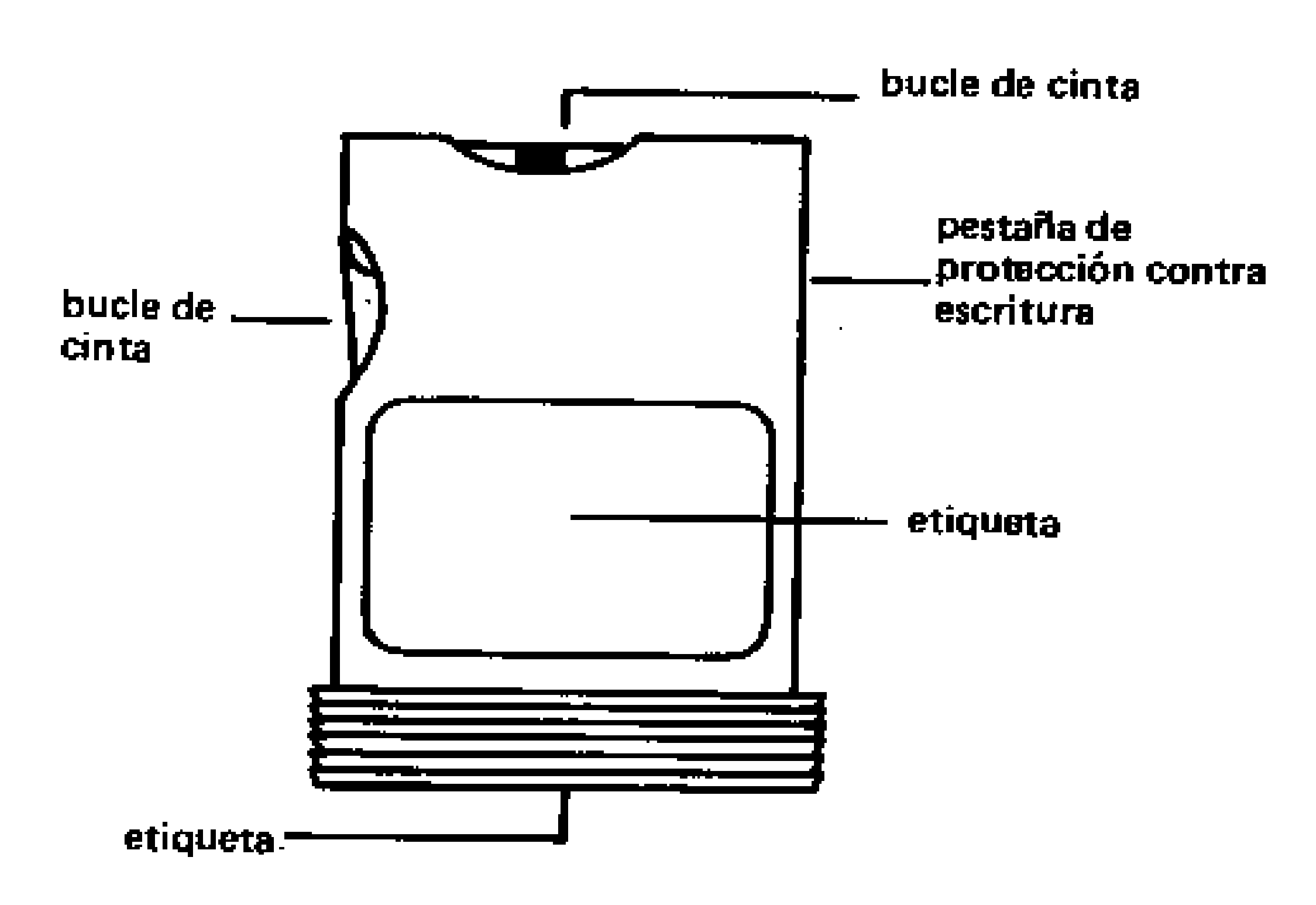


Fig. 11.3.– Esquema de un microdrive.

Como puede observarse, consta de un bucle de cinta magnética de alta calidad de una longitud de 200 pulgadas que se mueven sobre la cabeza de lectura/escritura del ordenador a una velocidad de 28 pul-

gadas por segundo de forma que se tardan 7 segundos y medio aproximadamente en realizar un bucle completo.

El fabricante recomienda tomar las siguientes precauciones durante el manejo de los microdrives:

- NO tocar nunca la cinta con los dedos o utilizar la funda protectora para cualquier uso que no sea el suyo propio.
- NO conectar o desconectar el ordenador con los microdrives colocados.
- Almacenar siempre los microdrives en sus fundas protectoras cuando no sean utilizadas.
- Introducir y sacar los microdrives de sus respectivos slots despacio y con cuidado.
- Asegurarse de que el microdrive está correctamente instalado en su slot antes de proceder a su utilización.
- No retirar el microdrive mientras esté siendo utilizado.

11.8. REDES DE COMUNICACION

Como indica la propia firma constructora, el Sinclair QL puede conectarse con otros QL's o Spectrum's hasta un máximo de 63 unidades, formando una red local llamada QLAN.

Cuando existan más de dos ordenadores conectados en esta red, cada uno de ellos debe poseer un *número de estación* prefijado de antemano con el comando NET.

El formato de este comando es:

NET [*d*] [_*s*]

donde *d:* indica la dirección de la transmisión según la codificación:

i – INPUT (entrada)
o – OUTPUT (salida)

s: indica el número de *estación* de trabajo y debe ser un número entero comprendido entre 0 y 127.

Cuando no se especifica el número de estación, se asume por defecto el 1.

La información que transita por una red de ordenadores conectados está particionada en bloques, de forma que la estación receptora de un bloque de información deberá suministrar a la estación emisora un reconocimiento positivo (ACK) de que dicha información ha llegado sin problemas y puede ser interpretada. En caso contrario, solicitará de la estación emisora el reenvío del mensaje mediante una petición de retransmisión.

La estación asignada al número 0 tomará las funciones de estación maestra y podrá controlar el flujo de información circulante por la red.

11.9. EL SISTEMA OPERATIVO QDOS

El QDOS es el nombre del sistema operativo del QL y es el encargado de supervisar las funciones del equipo entre las que cabe destacar el control de las tareas a realizar, las operaciones de Entrada/Salida para la pantalla y para el acceso a los microdrives, la comunicación vía red local o canales conectados, las entradas del teclado y la gestión de la memoria.

La figura 11-4 muestra un esquema de la memoria RAM del sistema dividida en secciones, cada una de las cuales cumple su propio objetivo dentro del sistema completo.

Las palabras SV_RAMT, SV_RESPR, etc., son las utilizadas por el constructor para designar direcciones dentro del área RAM del QL y poseen los siguientes significados:

SV_RAMT (RAM Top/Comienzo de la memoria RAM)
Que puede sufrir variación dependiendo de las sucesivas expansiones de memoria conectadas al ordenador.

SV_RESPR (Resident Procedures / Procedimientos Fijos)
Son aquellos procedimientos que se cargan al comienzo de la RAM. Una vez hecho esto, este espacio de memoria solamente puede ser modificado reinicializando el sistema. Estos procedimientos son tomados como extensiones del SuperBASIC.

SV_TRNSP (Transient Programs)
Cada uno de estos programas debe ser autosuficiente, esto es, contener y manejar sus propios datos e

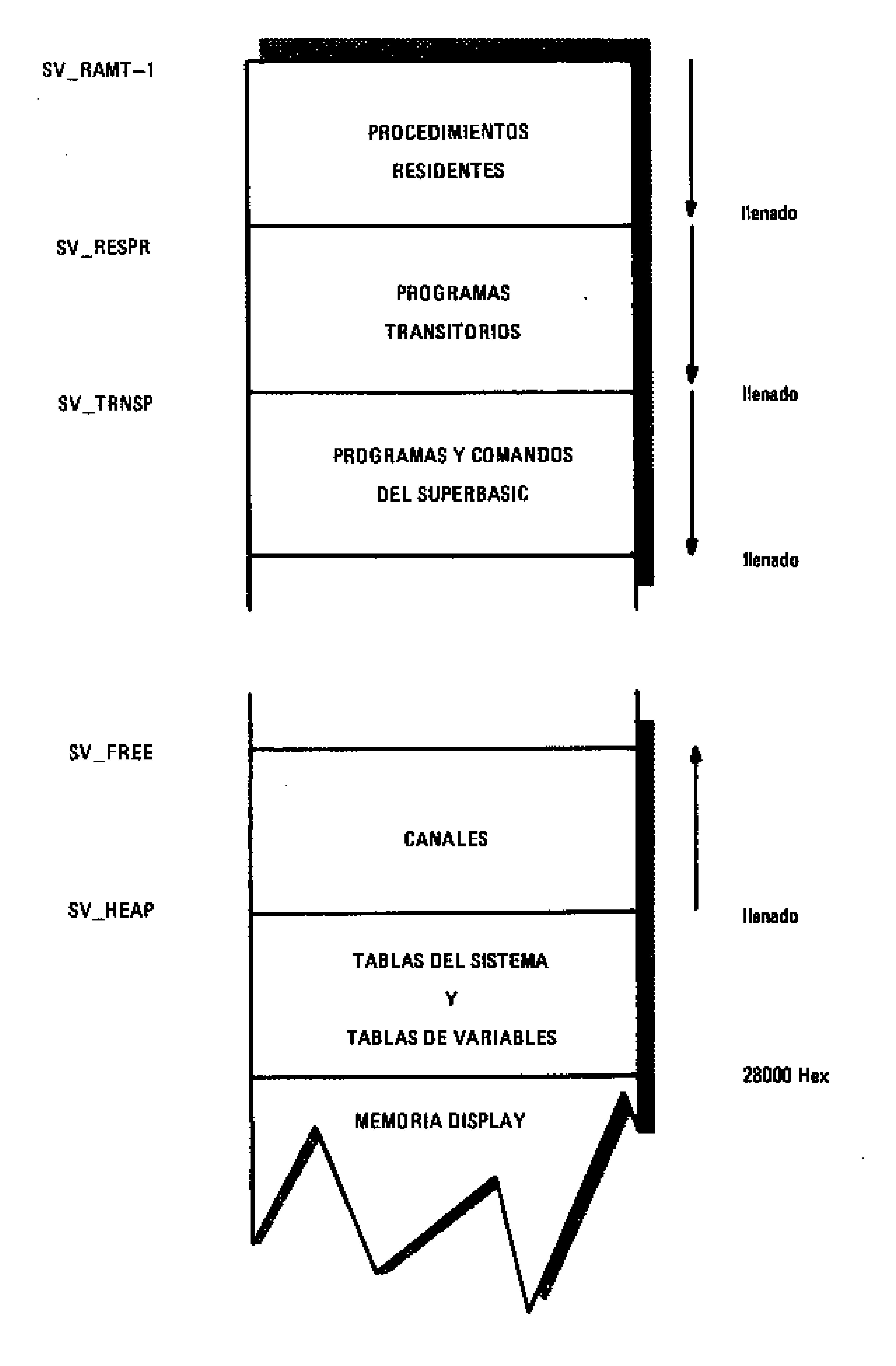


Fig. 11.4.— Estructura de la memoria RAM.

instrucciones. Estos programas son llamados a la ejecución mediante el comando EXEC.

SV_BASIC (Area destinada al SuperBASIC)
Que contiene todos los programas SuperBASIC cargados de forma que puede contraerse o expandirse en la medida de los requerimientos de tales programas.

SV_FREE (Area libre)
Se trata de un área libre de memoria que puede utilizar el QDOS para la creación de ficheros en microdrive desde la RAM.

SV_HEAP (System HEAP)
Se usa para almacenar definiciones de canal, etc., y para ser utilizado para áreas de trabajo en operaciones de Entrada/Salida.

SYSTEM Tables / System Variables
Inmediatamente anterior a la memoria direccionable de la pantalla para tablas y variables del sistema.

Cuando el sistema está procesando un comando del QDOS se dice que se encuentra en *modo supervisor* y no podrá realizarse otra función hasta que no se llegue al final de la ejecución de la llamada al supervisor.

Respecto de la multitarea podemos decir que el QDOS permite que un fichero pueda ser accedido por más de un proceso al mismo tiempo, de forma que estos ficheros pueden estar abiertos en modo exclusivo o en modo compartido.

En general, los trabajos dentro del control del QDOS pueden estar en un instante considerado, en uno cualquiera de los tres estados siguientes:

Activo: En disposición de ser ejecutado utilizando los recursos del sistema dependiendo de su propia prioridad de ejecución.

Suspendido: Se encuentra en disposición de ser ejecutado, pero a la espera de que se complete determinada operación de Entrada/Salida o bien otro trabajo cualquiera.

Inactivo: El trabajo posee la prioridad 0 y por tanto se encuentra inútil para su ejecución.

11.9.1. La instrucción CALL

Dentro de un procedimiento o programa en general pueden ejecutarse instrucciones escritas en código máquina directamente (juego de instrucciones del microprocesador 68008 de Motorola) mediante el uso del comando CALL cuya estructura es:

CALL *direcciones* [{ *,datos* }]

donde *direcciones:* representan direcciones con las que se llenarán los 6 registros generales de direcciones A0 a A5.
datos: representan datos con los que se trabajará y que estarán en los 7 registros generales para datos D1 a D7.

Este comando sólo podrá leer, por tanto, como máximo 13 parámetros.

Como el lector habrá observado, el registro de direcciones A6 no es utilizado por este comando.

Para regresar al SuperBASIC después de una serie de instrucciones máquina deberá escribirse:

```
MOVEQ # 0,D0
RTS
```

11.9.2. La instrucción CLEAR

El QL dispone de un comando llamado CLEAR que limpia el área de variables del SuperBASIC para el programa que esté en ejecución en ese momento, habilitando el espacio para el sistema operativo QDOS.

El formato del comando es simple:

CLEAR

Este comando puede utilizarse para acceder a un estado conocido dentro del SuperBASIC. Por ejemplo, la parada intencionada o accidental de un programa dentro de un procedimiento. Este comando hará que se devuelva el control al SuperBASIC, saliendo del ámbito del programa causante del fallo.

11.9.3. La función RESPR

El sistema operativo QDOS posee una función, la RESPR que permite reservar espacio de memoria residente para, por ejemplo, añadir más listas de procedimientos del SuperBASIC.

El formato de esta sentencia es:

RESPR *(memoria)*

donde *memoria:* debe ser una expresión-numérica (o una constante o una variable del mismo tipo) que señala el espacio de memoria reservado.

Ejemplo:

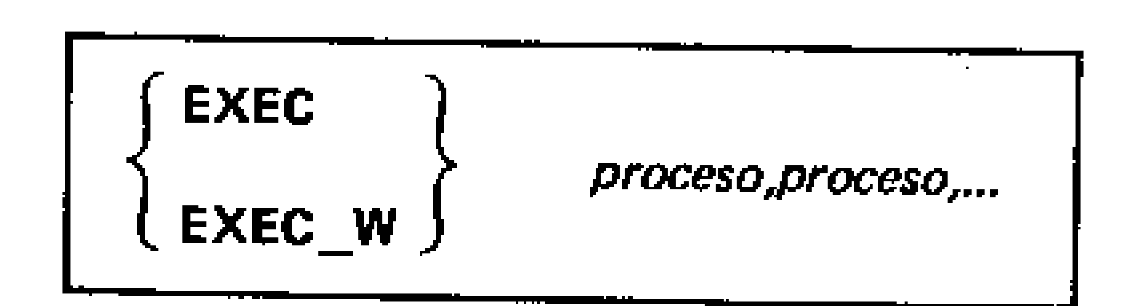
```
40   PRINT RESPR (1024)
```

que imprimirá la dirección de base de un bloque de 1024 bytes que se menciona como argumento en la propia función.

11.9.4. Multitarea: La instrucción EXEC

El Sistema Operativo QDOS del Sinclair QL dispone de la facilidad de ejecutar varios procesos simultáneamente. Esto se hace con los comandos EXEC y EXEC_W.

El formato de estos comandos es:



Cuando se utiliza la forma EXEC, el control se devuelve al procesador de comandos después de que todos los procesos implicados han comenzado su ejecución. Por contra, con el formato EXEC_W, espera hasta que todos los trabajos han terminado para devolver este control al procesador de comandos.

Ejemplos:

```
EXEC mdv1_agenda,mdv1_contable
EXEC_W mdv1_cuentas,datos,mdv1_impreso
```

Como ilustración de la función de multitarea se incluye el siguiente programa ejemplo que coloca en la pantalla un reloj de tiempo real que trabaja independientemente. Hay que ejecutar primero este programa sobre un microdrive colocado en mdv2.

Ejemplo:

```
100  REMark  Prueba de Multitarea
110  c = RESPR (100)
120  RESTORE
130  FOR i=0 TO 68 STEP 2
140        READ d : POKE_W i + c, d
150  END FOR i
160  SEXEC mdv2_ reloj, c, 100, 256
170  DATA  29439, 29697, 28683, 20033, 17402
180  DATA  48, 13944, 200, 20115, 12040
190  DATA  28691, 20033, 17402, 74, –27698
200  DATA  13944, 236, 20115, 8279, –11314
210  DATA  13944, 208, 20115, 16961, 16962
220  DATA  30463, 28688, 20035, 24794
230  DATA  0, 7, 240, 10, 272, 200
```

Una vez ejecutado este programa se tendrá en el microdrive el fichero "mdv2-reloj" y para ejecutarlo en modo multitarea basta con teclear:

```
EXEC mdv2_reloj
```

11.9.5. La instrucción SEXEC

Asimismo, el QDOS dispone de un comando que permite salvar un área de memoria de forma que sea susceptible de ser asignada y cargada con posterioridad y ejecutado con un comando EXEC.

Este comando es el llamado SEXEC y tiene el formato:

SEXEC *dispositivo,dirección,longitud,datos*

donde *dispositivo:* puede ser cualquiera de los que pueden conectarse al QL, ya sean microdrives o dispositivos colocados en puertas seriales.

dirección: representa el comienzo del área de memoria que va a ser salvada.

longitud: representa la longitud de la memoria que va a ser salvada, expresada en bytes.

datos: representa el espacio de datos, esto es, la longitud del área de datos que serán necesarios para la ejecución del programa.

Tanto *dirección*, *longitud* como *datos* pueden ser en el caso más general expresiones-numéricas (o constantes o variables numéricas).

Ejemplos:

SEXEC mdv1_programa,262164,3000,500
salvará las posiciones de memoria comenzando en la dirección 262164 con una longitud de 3000 bytes y un área de datos de 500 bytes.

11.10. EL SONIDO: La instrucción BEEP

El QL dispone de unas enormes facilidades para el manejo y generación de música o sonidos en general.

El SuperBASIC posee una instrucción, la BEEP para generación de sonidos.

El formato de la instrucción BEEP es:

BEEP *duración,pitch1,pitch2,grad_x,grad_y, wrap,distorsión,aleatorio*

donde *duración:* debe ser una cantidad comprendida entre –32768 y 32767 y especifica la duración del sonido en unidades de 72 microsegundos. Cuando se escribe una duración de 0, el sonido permanecerá activo hasta que sea cancelado por otra sentencia BEEP.

pitch1 y pitch2: especifica la altura tonal del sonido con dos posibles niveles simultáneos. Debe ser una cantidad comprendida entre 0 y 255, siendo 1 correspondiente a niveles altos y 255 a niveles bajos.

grad_x: Debe ser una cantidad comprendida entre –32768 y 15 y define el tiempo de intervalo entre frecuencias sucesivas.

grad_y: Define el tamaño de cada paso y debe ser un valor comprendido entre –8 y 7.

wrap: Se trata de un efecto especial que hace que el sonido "gire" un número concreto de veces. Debe ser una cantidad comprendida entre 0 y 32767.

distorsión: Define el nivel de distorsión o saturación del sonido y debe ser una cantidad comprendida entre 0 y 15.

aleatorio: Para efectos especiales. Comprendido entre 0 y 15.

Los dos únicos parámetros obligatorios de esta instrucción son la *duración* y el *pitch1*.

Asimismo, el SuperBASIC dispone de una función incorporada (BEEPING) que devuelve cero (0) si el ordenador no está sonando en el instante considerado y cualquier otra cosa (≠ 0) si está sonando.

Un ejemplo habitual para detener un sonido puede realizarse con las líneas de programa siguientes:

```
100 DEFine PROCedure silencio
110       BEEP
120 END DEFine
130 IF BEEPING THEN silencio
```

La característica de sonido dentro del QL es algo que debe experimentarse bastante antes de poder llegar a dominar todas sus posibilidades.

Incitemos desde aquí al lector para que lo haga así, probando cada uno de los parámetros que intervienen en el sonido.

Algunos ejemplos pueden ser:

```
BEEP 0,10,100,100,10
BEPP 0,33,77,12000,12
BEEP 0,10,100,2500,3,3
BEEP 20000,10,100,1000,10,5
```

Los siguientes valores de *pitch1* pueden ser tomados para la creación de tres octavas:

```
DATA 165,155,148,139,130,121,113,107,101,94,88
DATA 83,77,73,68,64,59,56,52,48,45,42,39,36
DATA 33,31,29,27,25,22,20,19,17,15,13,10
```

Mediante el comando BEEP pueden construirse alrededor de 72000 billones (con b) de posibilidades. ¡No intente el lector probar todas ellas el primer día!

APENDICE A

Los comandos del SUPERBASIC

A.1. INTRODUCCION

En este apéndice se estudian con detalle todos los comandos de uso que pueden ser ejecutados con los parámetros adecuados y con la ayuda del intérprete de SuperBASIC y el Sistema Operativo QDOS.

Estudiaremos los comandos NEW para creación de programas, RUN para su ejecución, AUTO y RENUM para numeración automática de líneas fuente, FORMAT para inicializar microdrives, SAVE para guardar programas o datos, DIR para visualizar el directorio de un microdrive, COPY para copiar datos o programas de un dispositivo a otro y un largo etcétera de todas las posibilidades que nos brinda el Sistema Operativo QDOS.

El lector puede compaginar la lectura de este apéndice con el resto del material del texto, ya que son materias perfectamente abordables por separado, necesitándose poseer ciertos conocimientos de comandos antes de poder trabajar con rigor en la construcción y ejecución de programas SuperBASIC.

A.2. CREACION DE PROGRAMAS: El comando NEW

Cuando se teclea este comando y cuya estructura es:

NEW

se borra automáticamente el contenido que posee la memoria del ordenador en ese momento, y ya está en condiciones de escribir un nuevo programa que se almacenará en la memoria de esta manera liberada.

Debe teclearse, por consiguiente, siempre antes de introducir cualquier programa nuevo que se desee se almacene en la memoria para, por ejemplo, ser salvado en microdrive posteriormente.

Este comando también limpia de contenido los canales o ventanas por defecto 0, 1 y 2.

A.3. EJECUCION DE PROGRAMAS: El comando RUN

La estructura de este comando es:

RUN [*expresión-numérica*]

Cuando se teclea este comando, el programa que en ese momento se encuentra en memoria pasa a ejecutarse.

Esta es pues la forma habitual de ejecutar programas cargados en la memoria del QL.

Si no se especifica *expresión-numérica,* el programa comenzará a ejecutarse desde la línea de numeración más baja; en caso contrario, cuando se especifica *expresión-numérica,* el programa comenzará a ejecutarse desde el punto mencionado.

Ejemplos:

RUN
comienza la ejecución desde la primera línea del programa.

RUN 100
comienza la ejecución en la línea 100

RUN 3 * 50
comienza la ejecución en la línea 150

A.4. NUMERACION AUTOMATICA DE LINEAS: EL COMANDO AUTO

Cada línea de un programa en SuperBASIC debe estar numerada y estos números de líneas del programa deberán ser ascendentes, aunque no importa el intervalo que exista entre una y la siguiente.

Con el fin de evitar al usuario la molestia de teclear dicho número como comienzo de cada línea puede utilizarse este comando cuyo formato es:

AUTO [*línea-comienzo,incremento*]

Cuando se utiliza únicamente el comando como:

AUTO

el sistema devolverá un 100 como número de la primera línea del programa y cada vez que se pulse la tecla ENTER se incrementará este valor en 10 unidades.

Ejemplo:

```
NEW
AUTO
100  REMark numera a partir de la 100
110  REMark de 10 en 10
120  PRINT "se acabo"
```

Si, por otro lado, deseáramos que la primera línea del programa comenzara en un número determinado (p. ej. 500) y en incrementos de, p. ej., 5 en 5 deberíamos escribir:

AUTO 500,5

Ejemplo:

```
NEW
AUTO 500,5
500  REMark numera a partir de la 500
505  REMark de 5 en 5
510  PRINT "se acabo"
```

Para suspender la acción del comando AUTO se pulsarán simultáneamente las teclas CTRL y la barra de espaciado.

A.5. EDICION DE UN PROGRAMA: El comando EDIT

Cuando se encuentra un programa en la memoria del ordenador puede tenerse acceso a alguna de las líneas del mismo, utilizando el comando:

EDIT *número-línea* [, *incremento*]

La *línea* mencionada aparecerá en la pantalla entonces, y será susceptible de ser modificada según la relación:

TECLA	Qué hace?
←	mueve un lugar a la izquierda
→	mueve un lugar a la derecha
CTRL y ←	borra el carácter a la izquierda
CTRL y →	borra el carácter del cursor

Para borrar una línea de un programa, basta con escribir el número de dicha línea y nada más. La línea desaparecerá automáticamente.

Para insertar una o varias nuevas líneas entre dos que ya existen, deberán numerarse las líneas a insertar en el orden que se desee se ejecuten y pasarán a ocupar su lugar automáticamente.

Cuando se menciona *incremento* en el comando se editarán sucesivamente las líneas que poseerán dicho incremento.

Ejemplos:

EDIT 100
editará la línea 100

EDIT 60,10
editará las líneas 60,70,80, etc.

A.6. INICIALIZACION DE MICRODRIVES: El comando FORMAT

Cuando Vd. adquiera un microdrive nuevo para utilizar con su QL, estará absolutamente vacío y sin inicializar. El primer paso para poder utilizar un microdrive será inicializarlo. Para este cometido se utilizará el comando FORMAT que posee la estructura siguiente:

FORMAT *dispositivo_nombre*

donde *dispositivo:* puede ser cualquiera, *mdv1* ó *mdv2*, según se encuentre nuestro microdrive en alguno de los slots mencionados.

nombre: será la denominación que ahora poseerá físicamente el microdrive y podrá ser utilizado posteriormente. Este nombre se rige por las reglas de formación de los identificadores en SuperBASIC, aunque en este caso no puede poseer una longitud de más de 10 caracteres. Es opcional.

Un microdrive puede ser formateado tantas veces como se desee. Debe advertirse, no obstante, que cuando se ejecuta el comando FORMAT, todas las informaciones que pudiera poseer (programas, datos, etc.) *serán automáticamente destruidas.*

Ejemplos:

```
FORMAT mdv1_cinta1
FORMAT mdv2_programas
```

Una vez formateado el microdrive aparecerá en pantalla el número de sectores disponibles para su uso en relación con el número total de sectores del dispositivo.

El fabricante recomienda proceder a formatear varias veces un microdrive antes de proceder a utilizarlo.

Esto puede hacerse según:

```
FOR i = 1 TO 5 : FORMAT mdv1....
```

A.7. SALVAGUARDA EN MICRODRIVE: El comando SAVE

Cuando se dispone de un programa en la memoria, una forma de perpetuarlo para posteriores usos es "salvándolo" en un microdrive. El comando que hace esto es el SAVE y posee la estructura:

SAVE dispositivo_nombre

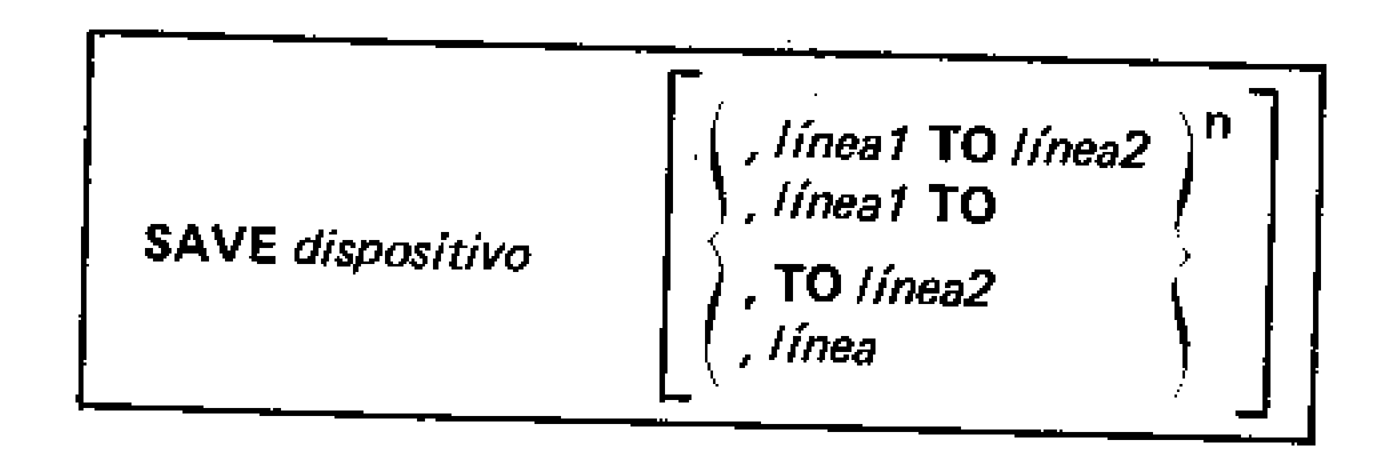
donde *dispositivo*: es bien *mdv1* ó *mdv2*,
nombre: es el nombre con el que se guardará el programa en el microdrive mencionado y el que deberá ser utilizado cuando se desee cargar de nuevo a la memoria para su ejecución.

Ejemplos:

SAVE mdv2_agenda
SAVE mdv1_grafico
etc.

No obstante, el comando SAVE dispone de algunas opciones que permiten salvar sólo ciertas partes de un programa en un dispositivo del QL.

El formato extendido del comando es:



donde *dispositivo:* es el mencionado con anterioridad.
línea TO línea2: salvará desde la *línea1* hasta la *línea2*. Ambas *línea1* y *línea2* pueden estar representadas por expresiones numéricas, constantes o variables numéricas.
línea1 TO: salvará desde la *línea1* hasta el final del programa.
TO línea2: salvará desde el principio del programa hasta la *línea2* mencionada.
línea: sólo salvará la *línea especificada.*

Ejemplos:

SAVE mdv2_agenda, 20 **TO** 80
salvará las líneas 20 a la 80 del programa mdv2_agenda

SAVE mdv1_programa, 10,20,40
salvará las líneas 10,20 y 40 del programa

SAVE SER1
salvará el programa entero que se encuentre en memoria en el canal en serie 1.

A.8. VISUALIZACION DEL DIRECTORIO DE UN MICRODRIVE: El comando DIR

Dentro de un microdrive pueden existir programas, ficheros, datos para proceso de textos, gráficos, etc.

Cuando se desee visualizar el contenido de un microdrive, deberá ejecutarse el comando DIR cuya estructura es:

DIR *dispositivo_nombre*

donde *dispositivo:* es *mdv1* ó *mdv2* según dónde esté alojado el microdrive cuyo directorio se desea visualizar.
nombre: es el nombre físico que posee el microdrive y que ha debido especificarse en su momento con el comando FORMAT. Este nombre puede omitirse.

Ejemplos:

DIR mdv1_cinta1
DIR mdv2_graficos
DIR mdv1_
etc.

La ejecución del comando DIR sobre un microdrive proporciona en pantalla una copia del directorio de ese dispositivo con el número de sectores disponibles respecto del número de sectores totales.

El formato de salida de estas informaciones es:

Nombre Volumen
sectores-libres/sectores-totales **sectors**
nombre-fichero-1
....
nombre-fichero-n

Cuando este directorio posee muchos nombres de ficheros, puede detenerse la visualización en la pantalla de dichos nombres tecleando CTRL y F5 simultáneamente.

A.9. COPIA DE PROGRAMAS O FICHEROS: El comando COPY

Una vez que un programa o fichero es salvado a microdrive con el comando SAVE, puede copiarse a otro lugar con el propósito más generalizado de poseer una copia de seguridad, o para otros usos.

El formato del comando COPY es:

{ **COPY** / **COPY_N** }	*dispositivo_nombre1* **TO** *dispositivo_nombre2*

donde *nombre1* y *nombre2* son los identificadores del original y de la copia respectivamente.

Como siempre, *dispositivo* puede ser bien *mdv1* o *mdv2*.

Ejemplos:

COPY *mdv1_agenda* **TO** *mdv2_diario*

El comando COPY, en resumen, copia un fichero desde un dispositivo de entrada hasta un dispositivo de salida. La versión COPY_N del comando, copia el contenido del fichero al igual que el anterior, pero *no copiará* la cabecera asociada al fichero. Estas cabeceras podemos identificarlas con los dispositivos que poseen directorios (por ejemplo, los microdrives) y el comando COPY_N debe utilizarse cuando se quieran copiar ficheros a dispositivos exentos de tales cabeceras.

Ejemplo:

COPY_N mdv1_programa **TO SER1**
que copiara el fichero mencionado a la puerta en serie 1 obviando las informaciones de cabecera.

A.10. SUPRESION DE PROGRAMAS O FICHEROS: El comando DELETE

El comando DELETE se utilizará cuando se desee borrar de un determinado microdrive algún fichero o programa.

El comando DELETE no suprime propiamente todo el fichero, sino únicamente la referencia al mismo, que se encuentra en el directorio del microdrive.

El formato del comando es:

DELETE *dispositivo_nombre*

donde *dispositivo:* es como siempre, bien *mdv1* o bien *mdv2*.
nombre: es el identificador del programa o fichero que se desea borrar.

Ejemplos:

DELETE mdv1_agenda
borrara el programa "agenda" del directorio del microdrive colocado en *mdv1*.

Una vez ejecutado el comando DELETE sobre algún fichero de un microdrive en la ejecución posterior del comando DIR no aparecerá ya tal fichero.

A.11. CARGA DE PROGRAMAS: El comando LOAD

Este comando posee la función inversa al comando SAVE. Cuando se escribe LOAD y el nombre de programa, este se copiará –si existe–

del microdrive a la memoria del QL de forma que se encuentre listo para su ejecución.

El formato del comando es:

LOAD *dispositivo_programa*

donde *dispositivo:* es *mdv1* o bien *mdv2*.
programa: es el nombre del programa con el que está registrado en el directorio del microdrive de que se trate.

Ejemplo:

LOAD mdv1_agenda
cargara en la memoria el programa "agenda" que debera estar salvado previamente en el microdrive situado en *mdv1*

Ha de hacerse notar que cuando un programa está siendo cargado en memoria mediante un comando LOAD y se detecta un error sintáctico de SuperBASIC, el sistema introducirá la palabra

MISTAKE

entre el número de línea de la instrucción y la sentencia propiamente dicha.

La ejecución posterior de un programa con este tipo de circunstancias conllevará la aparición de un error de ejecución.

La ejecución de un comando LOAD lleva implícita la ejecución anterior de un comando NEW que no es necesario teclear.

A.12. VISUALIZACION DE UN PROGRAMA: El comando LIST

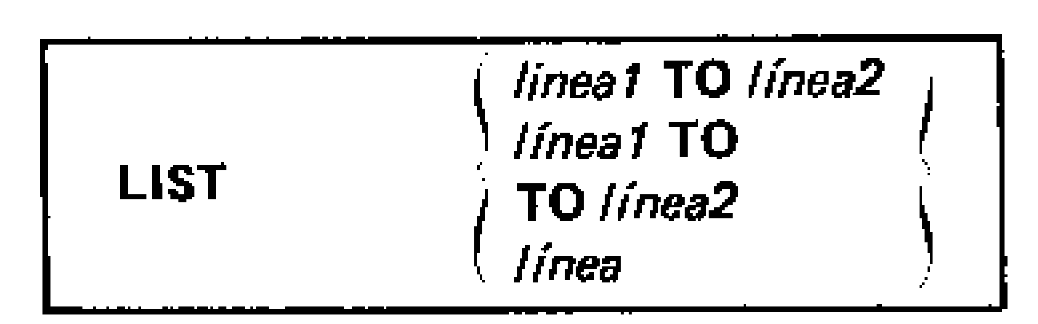
Con el comando LIST se pueden visualizar en pantalla ciertas instrucciones de las que está formado un programa previamente cargado en memoria.

El formato más general del comando es:

LIST

no hace falta especificar el nombre del programa. Listará, por tanto aquél que en ese momento se halle en la memoria.

No obstante, el comando LIST puede poseer otras apariencias:



donde *línea1 TO línea2:* listará todas las líneas del programa comprendidas entre línea1 y línea2.
línea1 TO: listará desde la línea especificada (línea1) hasta el final del programa.
TO línea2: listará desde el principio del programa hasta la línea especificada (línea2).
línea: listará exclusivamente la línea especificada.

La visualización de las líneas de un programa puede derivarse hacia un canal específico (p. ej. una impresora) que tendrá que mencionarse como el primero de los parámetros de este comando.

Ejemplos:

LIST
listará todo el programa completo

LIST 10 TO 250
listará todas las líneas desde la 10 a la 250 ambas inclusive.

LIST #6,100 **TO** 200
listará en el canal especificado el rango de líneas descrito.

Como siempre, un comando LIST puede cancelarse apretando las teclas CTRL y la barra espaciadora.

A.13. BORRADO DE LINEAS DE UN PROGRAMA: El comando DLINE

Con el comando DLINE pueden borrarse una o un rango de líneas de un programa SuperBASIC.

El formato de este comando es:

DLINE	*línea1* **TO** *línea2* *línea1* **TO** **TO** *línea2* *línea*

donde *línea1 TO línea2:* borra todas las líneas comprendidas dentro del rango mencionado.
línea1 TO: borra de la línea especificada *(línea1)* hasta el final del programa.
TO línea2: borra desde el principio del programa hasta la línea especificada *(línea2)*
línea: borra la línea especificada.

Estos cuatro procedimientos pueden incluirse en un solo comando DLINE separados por comas.

Ejemplos:

DLINE 20 **TO** 50,80,100 **TO** 150
borrará: – las líneas 20 a la 50
– la línea 80
– las líneas 100 a la 150

DLINE
no borrara nada

A.14. RENUMERACION DE LINEAS: El comando RENUM

Este comando permite la renumeración de todo o parte del programa que se encuentra en memoria en ese momento.

El formato de este comando es:

RENUM [*línea1* [**TO** *línea2;*] [*número*] [*,incremento*]]

cuando el comando se ejecuta sin opciones, entonces se renumera todo el programa, empezando por la línea 100 y en incrementos de 10.

Cuando se especifican opciones, esto significa:

línea1: número de la línea de comienzo
línea2: número de la línea final
número: con el que se empezará a renumerar el rango de líneas anterior.
incremento: diferencia entre dos líneas consecutivas.

Ejemplos:

RENUM
renumera todo el programa empezando en la primera línea con el número 100 y en incrementos de 10.

RENUM 150;250
renumera desde la 150 a la 250 en incrementos de 10.

A.15. CARGA Y EJECUCION DE PROGRAMAS: El comando LRUN

Este comando posee las mismas funciones que los LOAD y RUN combinados. Es decir, primero carga en la memoria del ordenador el programa mencionado (LOAD) y acto seguido lo ejecuta (RUN).

El formato del comando es:

LRUN *dispositivo_programa*

donde *dispositivo:* es o bien *mdv1* o *mdv2.*
programa: es el identificador del nombre de programa que ha de ser cargado y ejecutado.

Ejemplos:

LRUN mdv1_agenda

LRUN mdv2_grafico

etc.

A.16. CANCELACION DE COMANDOS

Siempre puede detenerse la ejecución de un programa o de un comando presionando la tecla:

CTRL y después manteniendola apretada **barra**

Cualquier programa o comando que sea detenido de esta manera puede volver a ser reejecutado desde el punto de su interrupción utilizando el comando:

CONTINUE

A.17. LIMPIEZA DE LA PANTALLA: El comando CLS

Como ya habíamos pergeñado en el capítulo 9, el comando CLS se utiliza para limpiar la ventana (window) por defecto en la ejecución, o bien cualquiera otra ventana asociada a un canal concreto. Este comando no afecta al borde (BORDER) de la ventana –si la tuviera– que permanecerá inalterable.

El formato del comando es:

CLS [*canal,*] [*fragmento*]

donde *canal:* representa el número del canal de la ventana asociada que se desea limpiar de cualquier contenido.
fragmento: da la posibilidad de limpiar una cierta parte de la ventana elegida según la relación:

0 – la pantalla entera
1 – la parte superior excluyendo la línea del cursor
2 – la parte inferior excluyendo la línea del cursor
3 – la línea del cursor
4 – la línea del cursor a partir de la propia posición del mismo.

Ejemplos:

CLS
Limpia la pantalla entera

CLS 3
limpia la línea del cursor

CLS #2,1
limpia la parte superior de la ventana 2 asociada a tal canal.

A.18. FUSION DE PROGRAMAS: El comando MERGE

El comando MERGE se utiliza para fusionar y entremezclar las líneas de varios programas.

El formato del comando MERGE es:

MERGE *dispositivo_nombre*

donde *dispositivo:* es o bien *mdv1* ó *mdv2*
nombre: es el nombre del programa que se fusionará con el que se encuentre actualmente en memoria.

Supongase que se tienen en el dispositivo *mdv1* los dos programas *prog1* y *prog2* como:

prog1

```
10  REMark prog1
20  FOR i= 1 TO 10
30      PRINT i
40  NEXT i
50  PRINT "final prog1"
60  STOP
```

prog2

```
15  REMark prog2
25  PRINT "valores de i";
50  PRINT "final prog2"
```

Para fusionar en uno sòlo las instrucciones de ambos programas deberá escribirse:

```
LOAD  mdv1_prog1
```

que cargará en memoria el prog1.

Y después

```
MERGE mdv1_prog2
```

las instrucciones de *prog2* serán fusionados con las de *prog1* ya en memoria, quedando como resultado:

```
10   REMark prog1
15   REMark prog2
20   FOR i = 1 TO 10
25        PRINT "valores de i"
30        PRINT i
40   NEXT i
50   PRINT "final prog2"
60   STOP
```

Como puede observarse, las líneas de ambos programas han sido fundidas de acuerdo con sus numeraciones respectivas.

Obsérvese que ambos programas PROG1 y PROG2 poseen una línea con un número común (la 50). Nótese que la correspondiente a *prog2* ha sustituido o "machacado" a la de *prog1*.

Debe tenerse especial cuidado con verificar que las numeraciones de ambos programas no sean coincidentes.

La fusión de programas es bastante útil cuando se trata de traspasar subrutinas de un programa a varios que utilicen el mismo procedimiento o función.

A.19. FUSION Y EJECUCION DE PROGRAMAS: El comando MRUN

Con el comando MRUN se fusionará el fichero que se mencione en la instrucción (que será tratado como un programa) con el que actualmente se encuentre en memoria y se ejecutará inmediatamente.

El formato del comando es:

MRUN *nombre*

donde *nombre:* representa el fichero (programa) residente en un microdrive que se especifica.

Ejemplos:

MRUN mdv1_partebajas
fusionara el fichero *partebajas* con el actual cargado en memoria y lo ejecutará inmediatamente.

MRUN mdv2_agenda
fusionara el fichero *agenda* (cargado en el microdrive 2) con el programa actualmente en memoria y pasará a ejecutarlo.

A.20. LOS CANALES (CHANNELS)

Como ya habíamos visto en el apartado de las *ventanas* (windows), un canal es un medio de comunicación para entrada o salida de datos desde o hacia un dispositivo cualquiera del QL.

Antes de que un canal pueda ser usado debe ser *abierto* (comando OPEN) para tales usos. No obstante, existen ciertos canales del QL que no necesitan abrirse, como son el teclado y la pantalla.

En general, cuando se ha dejado de utilizar un canal, deberá procederse a cerrarlo (comando CLOSE).

Ejemplo:

Abrir un canal de la *puerta en serie* SER1.

```
OPEN #5, SER1
```

Para cerrar este canal después de su uso basta con escribir:

```
CLOSE #5
```

sin necesidad ya de mencionar el nombre asociado.

En el capítulo 10 dedicado a ficheros, un canal puede estar también perfectamente asociado con un fichero. Los datos pueden ser leídos utilizando la instrucción INPUT asociada con un canal o bien pueden leerse datos directamente desde un canal, un carácter cada vez, mediante el uso de INKEY$. Caso análogo cabe decir de las instrucciones de escritura sobre canales mediante el uso de sentencias PRINT asociadas con números de canal.

APENDICE B

Los microprocesadores del QL

B.1. INTRODUCCION

En este apéndice se presenta una breve visión de la arquitectura interna y funcionamiento de los dos microprocesadores principales del Sinclair QL: el 8049 de Intel y el 68008 de Motorola.

El primero en describirse es el I8049 donde se expone al lector no experimentado en microprocesadores, la estructura completa de dicho chip que contiene, como veremos, en una sola pastilla, todo un ordenador completo, incluyendo memorias ROM y RAM y Unidades de Control y de Aritmética y Lógica. Este procesador efectúa en el QL misiones secundarias de apoyo, como pueden ser el manejo de sonido, el teclado, etc.

Acto seguido, se presenta con más detalle el microprocesador principal del QL: el 68008 de Motorola, encargado de las funciones principales del ordenador. Se estudian los conceptos de *bus de direcciones* y *bus de datos* y todo ello complementado con gráficos y tablas clarificadoras. No obstante, y dada la gran profundidad del tema tratado en este apéndice se recomienda al lector consultar la bibliografía citada al respecto para mayor detalle.

B.2. EL MICROPROCESADOR 8049

Como ya veremos en las siguientes líneas, el microprocesador 8049 de Intel es en esencia un microordenador de 8 bits en una única pastilla o chip. Si bien es cierto que no es tan potente como su compañero el 68008 de Motorola, mantiene por sí mismo una cierta complejidad

de cálculo y posee sus propias memorias de sólo lectura (ROM) y de acceso aleatorio (RAM).

En la figura B-1 mostramos al lector una variante de la figura 1-1 vista en el primer capítulo de este libro, donde pueden observarse los constituyentes básicos necesarios de un ordenador, e incluso de un microprocesador.

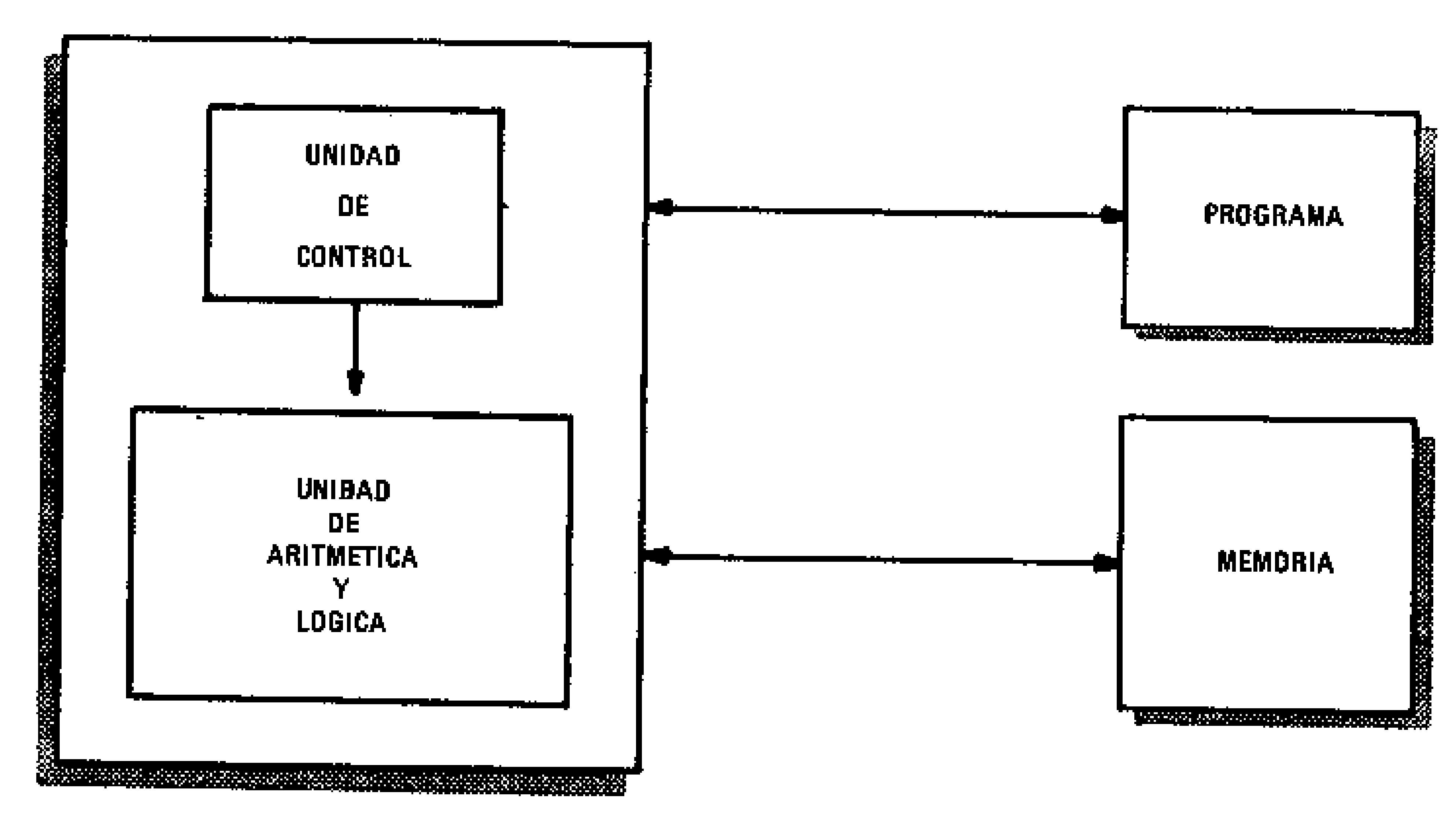


Fig. B.1.– Los componentes básicos de un ordenador.

Lo primero que se hace necesario en todo microprocesador es una unidad de cálculo a la que habitualmente se llama *Unidad Aritmética y Lógica (UAL)*. Como su propio nombre indica y como ya apuntábamos en el capítulo primero, esta unidad es la encargada de realizar operaciones de tipo aritmético y lógico. Se trata pues de una unidad absolutamente imprescindible.

Para realizar sus cálculos, la UAL necesita de un control, de un programa director, que deberá estar en la *Unidad de Control*. Por supuesto, el microprocesador necesita de algún lugar donde se encuentre almacenado el código de la instrucción para que, una vez decodificada, se efectúe la operación correspondiente.

Del mismo modo, la Unidad de Control, deberá controlar el tráfico de informaciones del microprocesador con los restantes elementos del ordenador, como puedan ser las áreas de memoria y las puertas de entrada/salida.

Dada la estrecha relación que guardan entre sí, a la conjunción de la Unidad Aritmética y Lógica y la Unidad de Control, se le suele llamar *Unidad Central de Proceso (UCP)* aunque es frecuente que ambas unidades se encuentren físicamente separadas.

A las instrucciones que son ejecutadas por la UCP directamente se las conoce con el nombre de *"programa"*. En el caso del microprocesador 8049 el programa no es otra cosa que una serie de instrucciones máquina que deberán ser decodificadas por la Unidad de Control, de forma que se obtengan operaciones del tipo que sea para ejecutar en la UAL.

Todo lo anterior debe ser complementado con un almacenamiento de datos que deberán incluir memoria para los resultados intermedios de los programas o bien como un almacenamiento habitual de datos. Así, tenemos la *Memoria* del microprocesador, que podrá contener bien las instrucciones del programa, bien los datos o bien ambas cosas. No obstante, para nuestros propósitos será conveniente que distingamos entre dos tipos de memoria.

El microprocesador 8049 de Intel es un ordenador completo en un solo chip, esto es, posee UAL, Unidad de Control, posee un programa y una memoria para datos y todo ello contenido dentro de un único chip físico.

Generalmente, un chip de este tipo consta de una pastilla de silicona recubierta de una protección cerámica o plástica que permite al microprocesador comunicarse con el exterior mediante impulsos eléctricos. A lo largo de los dos lados mayores de la pastilla se encuentran las conexiones eléctricas (llamadas *pins*) que serán las puertas de salida y entrada al exterior del microprocesador. Normalmente existen dos filas de pins situados a lo largo de los lados mayores del paralelepípedo del chip por lo que se les suele llamar *DIP's (dual in-line packages)*.

El microprocesador 8049 de Intel posee 40 pins y es por esta razón por la que se le llama chip "40 DIP".

La figura B-2 muestra la arquitectura o estructura del 8049. Veremos solamente alguna de las características más importantes de este microprocesador.

Observe el lector en la figura, primero que nada, como están dispuestos los cuatro elementos básicos de todo ordenador dentro del chip. Se posee una UAL de 8 bits estrechamente ligada con un acumulador también de 8 bits. La Unidad de Control no está descrita propiamente en la figura anterior, pero debe hacerse notar que sí se encuentra físicamente dentro del propio chip como sería de esperar.

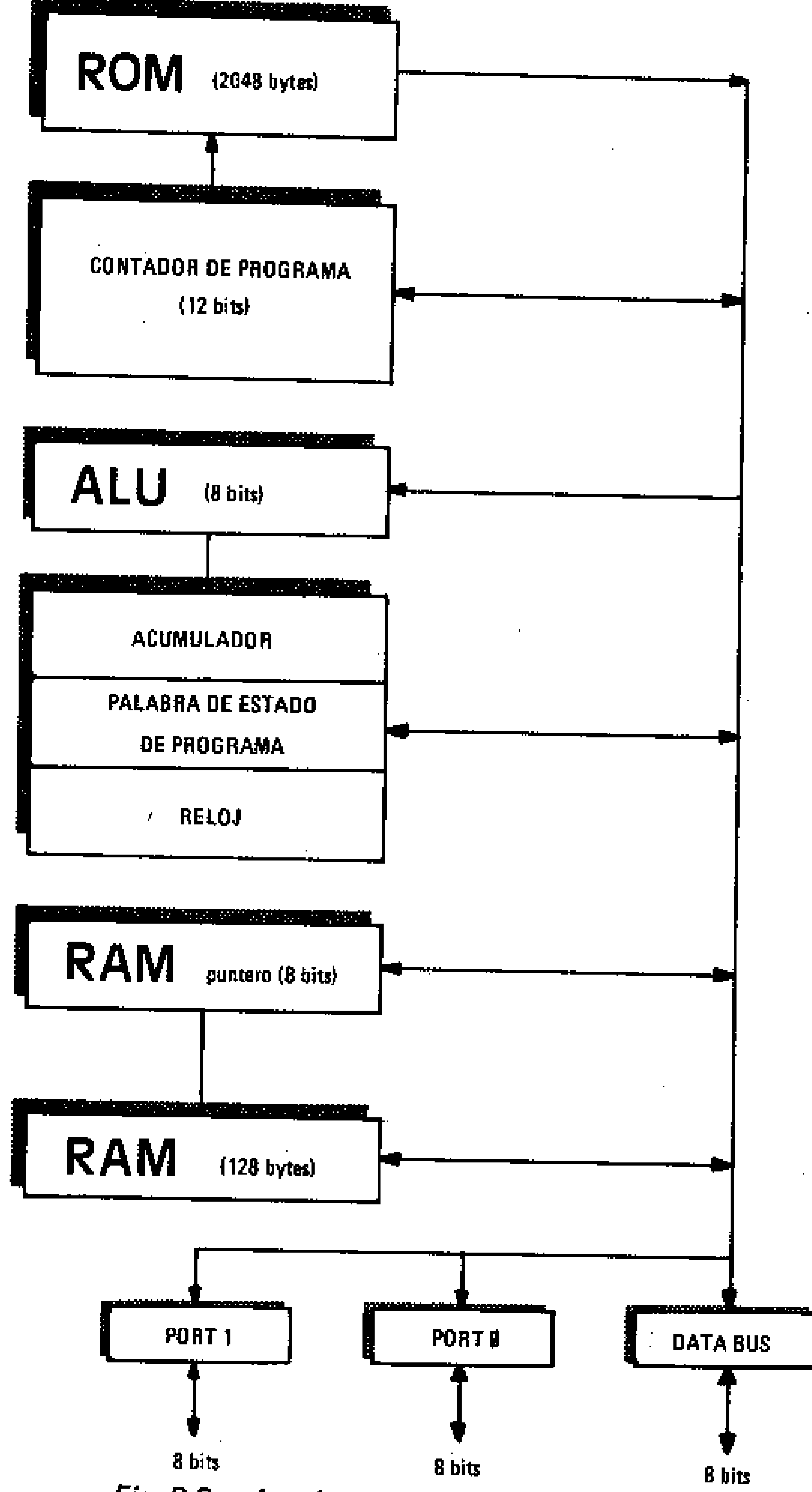


Fig. B.2.– Arquitectura del 8049 de INTEL.

El tercer elemento, el programa, se encuentra contenido en la sección ROM (Read Only Memory) que consta de un conjunto de direcciones de contenido fijo e inalterable. La memoria ROM de un ordenador se usa principalmente para almacenar programas que luego serán ejecutados por el sistema o bien programas de usuario que deberán ser traducidos a instrucciones máquina (como por ejemplo, programas en SuperBASIC).

Dentro del QL se encuentra precisamente en memoria ROM el intérprete de SuperBASIC que se encarga de recoger el programa escrito por el usuario y situado en la memoria RAM y de convertirlo a instrucciones que pueden ser directamente ejecutadas por la UCP.

Esta forma de almacenar permanentemente el intérprete de BASIC en memoria ROM es habitual de forma que no pueda modificarse accidentalmente su contenido.

El programa de control para el 8049 se encuentra almacenado en ROM en las 2 K's disponibles en el chip. Este programa consta de un conjunto de instrucciones que controlan la forma de operar de los distintos dispositivos de entrada/salida conectables de forma que el microprocesador principal del QL, el 68008 de Motorola puede concentrar toda su potencia en la ejecución del programa de usuario. Desde luego, en algunas ocasiones, el programa del 68008 deberá esperar a que el 8049 complete una determinada operación antes de actuar de nuevo, especialmente todas aquellas informaciones provenientes del teclado.

En la figura B-2 puede observarse como existen posibles salidas del 8049 para el control de dispositivos de entrada/salida (E/S) y que son la *Puerta-1, Puerta-2* y el *bus de datos.*

Las puertas de E/S son las que se conectan directamente a los dispositivos de E/S y el bus de datos puede comunicarse con dichos dispositivos a través de direcciones de memoria. Todas las puertas y el bus de datos son de 8 bits, aunque cuatro bits de la puerta-2 pueden utilizarse para conectar un microprocesador de E/S expansor 8243 de Intel.

Evidentemente existen multitud de ventajas en los ordenadores que utilizan dos microprocesadores, uno para los dispositivos de E/S y el otro para el proceso propiamente dicho frente a los tradicionales ordenadores personales mono-chip encargado de gestionar todos los recursos del sistema con la consiguiente depreciación de la memoria y tiempos de proceso.

El almacenamiento de los datos en el 8049 se realiza en una memoria RAM de acceso directo de 128 bytes (octetos). Este tipo de

memorias RAM (Random Access Memory) poseen la característica de que los datos almacenados en ellas pueden ser modificados en el momento oportuno. De esta forma, en el instante de conectar el QL, las instrucciones del programa de control se hallan en ROM, mientras que la memoria RAM se encuentra vacía de contenido.

Desde luego, la memoria RAM del 8049 de 128 bytes puede parecer excesivamente limitada, pero es perfectamente válida para el tipo de operaciones que va a efectuar dicho microprocesador.

El Intel 8049 posee unas direcciones especiales de memoria llamadas "registros" que poseen la misión especial de trabajar y de realizar operaciones con valores de datos.

Las primeras 8 direcciones de la memoria RAM la forman un conjunto de 8 registros, mientras que el segundo conjunto de registros componen la memoria RAM de usuario, esto es, direcciones que ya no poseen un propósito especial para realizar determinadas funciones.

El conjunto de instrucciones del 8049 es bastante sencillo y limitado y está diseñado especialmente para trabajar con pequeñas cantidades de memoria. Así, por ejemplo, la memoria del programa de control (ROM) se encuentra dividida en páginas de 256 bytes y para acceder de una página a otra es necesaria la ejecución de una instrucción especial de bifurcación del repertorio de instrucciones.

Tanto la memoria ROM como la RAM pueden ser expandidas añadiendo módulos exteriores de ampliación con el inconveniente añadido de que el proceso indefectiblemente se hará más lento. A mayor extensión de la memoria, mayor número de saltos o bifurcaciones.

Dado que el registro especial de *contador de programa* (ver figura B-2) posee 12 bits, la máxima extensión que podrá poseer la memoria ROM es de 4 kbytes, de forma que como el 8049 ya posee 2 Kbytes en ROM, esta posible adición de memoria adicional no podrá superar los 2 Kbytes.

B.3. EL MICROPROCESADOR 68008

El microprocesador 68008 de Motorola es el que ha sido utilizado por Sinclair Research para implementar la mayor parte de las funciones de proceso del QL.

El MC68008 es un microprocesador *contador de programa* erna de registros de 32 bits, aunque posee un bus de datos de sólo 8 bits. Veámos estos aspectos por separado.

El Bus de Direcciones

Como ya sabemos del microprocesador 8049 de Intel estudiado con anterioridad, el *bus de direcciones* se emplea para direccionar a distintos espacios de memoria. En el caso del MC68008 y dado que la memoria RAM se encuentra fuera del propio ámbito del chip, este bus de direcciones es absolutamente necesario.

Como todo microprocesador, posee el *contador de programa* que contiene en todo momento la dirección de la instrucción del programa que está siendo ejecutada. Cuando este registro posea solamente 16 bits, el número máximo de posibles direccionamientos es de 64 Kbytes. Cada uno de los bits de los que está compuesto el registro contador de programa se corresponde con una conexión eléctrica a una patilla (pin) del propio microprocesador con la memoria utilizada (ROM y RAM).

El Bus de Datos

El bus de datos se utiliza para la transferencia de datos de la memoria al microprocesador. En cualquier caso, el padre de la familia de estos microprocesadores, el MC68000 posee un contador de programa de 32 bits, de los que solamente 23 son utilizados como bits de direcciones.

El MC68008 del QL posee un bus de datos de 8 bits, por consiguiente el número de conexiones utilizadas para este propósito en el chip se verá reducida a esta cantidad. Dado que pueden enviarse operaciones con 16 bits simultáneamente, podemos decir que el MC68008 es algo más lento que su hermano mayor el MC68000, pero mucho más rápido que los procesadores normales de 8 bits, pudiendo decir que se trata realmente de un procesador de 16 bits. El bus de direcciones del MC68008 posee 20 bits de forma que es susceptible de direccionar una memoria de 1 Mbyte.

La figura B-3 muestra un diagrama comparativo entre los microprocesadores MC68000 y MC68008.

Los pins etiquetados como A0 a A19 del MC68008 corresponden al bus de direcciones (cada patilla es un bit). El bus de datos se tiene con los pins D0 a D7.

El microprocesador 68008 es un chip estructurado donde aparece el concepto de *estados privilegiados*. Este procesador solamente puede trabajar según dos estados privilegiados: el estado de *usuario* y el estado de *supervisor*.

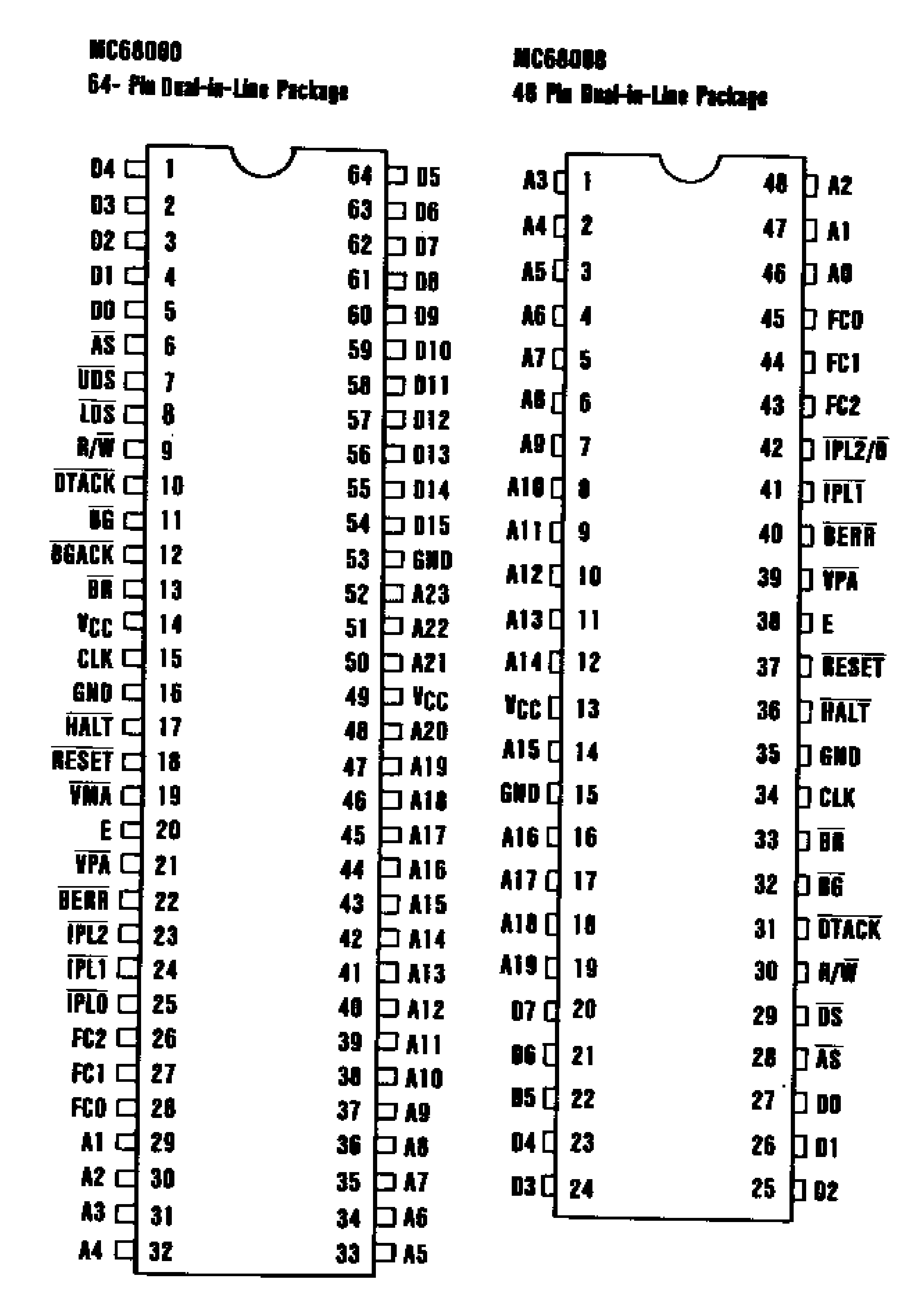


Fig. B.3.— Esquemas de los microprocesadores 68000 y 68008 de Motorola.

En cada uno de estos estados de privilegio solamente pueden ser ejecutadas ciertas operaciones o instrucciones, mientras que las restantes quedan *enmascaradas.*

Este sistema de privilegio en las operaciones es la piedra angular para el desarrollo de la multitarea dentro del QL, de forma que varios programas que estén siendo ejecutados simultáneamente, no se vean interferidos unos con otros. Así pues, todas las funciones del QDOS vistas en el capítulo 11 se encuentran dentro del estado supervisor en modo privilegiado, siendo el de mayor prioridad.

El lector interesado en profundizar sobre el tema puede consultar la bibliografía que se acompaña.

La arquitectura interna del MC68008 es idéntica a la dél MC68000 y posee 16 *registros* de 32 bits, dos *punteros de pila* de 32 bits, un *contador de programa* de 32 bits y un *registro de estado* de 16 bits.

La figura B-4 muestra la arquitectura de las instrucciones susceptibles de ser ejecutadas con el microprocesador 68008.

El primer conjunto de registros mostrado en la figura B-4 muestra 8 registros de datos (del D0 al D7) que permiten operaciones con un byte (8 bits) con una palabra (16 bits) o con una palabra-larga (32 bits). Seguidamente se muestran 7 registros de direcciones (del A0 al A6) que también pueden ser utilizados como registros índice.

El registro A7 muestra las dos pilas de uso según los estados de privilegio de usuario y de supervisor. En la parte inferior de la figura se dibujan los registros del programa y de estado.

La tabla de la figura B-5 muestra el repertorio de instrucciones del microprocesador y con ello damos por terminado este breve repaso a la arquitectura interna de los procesadores del QL.

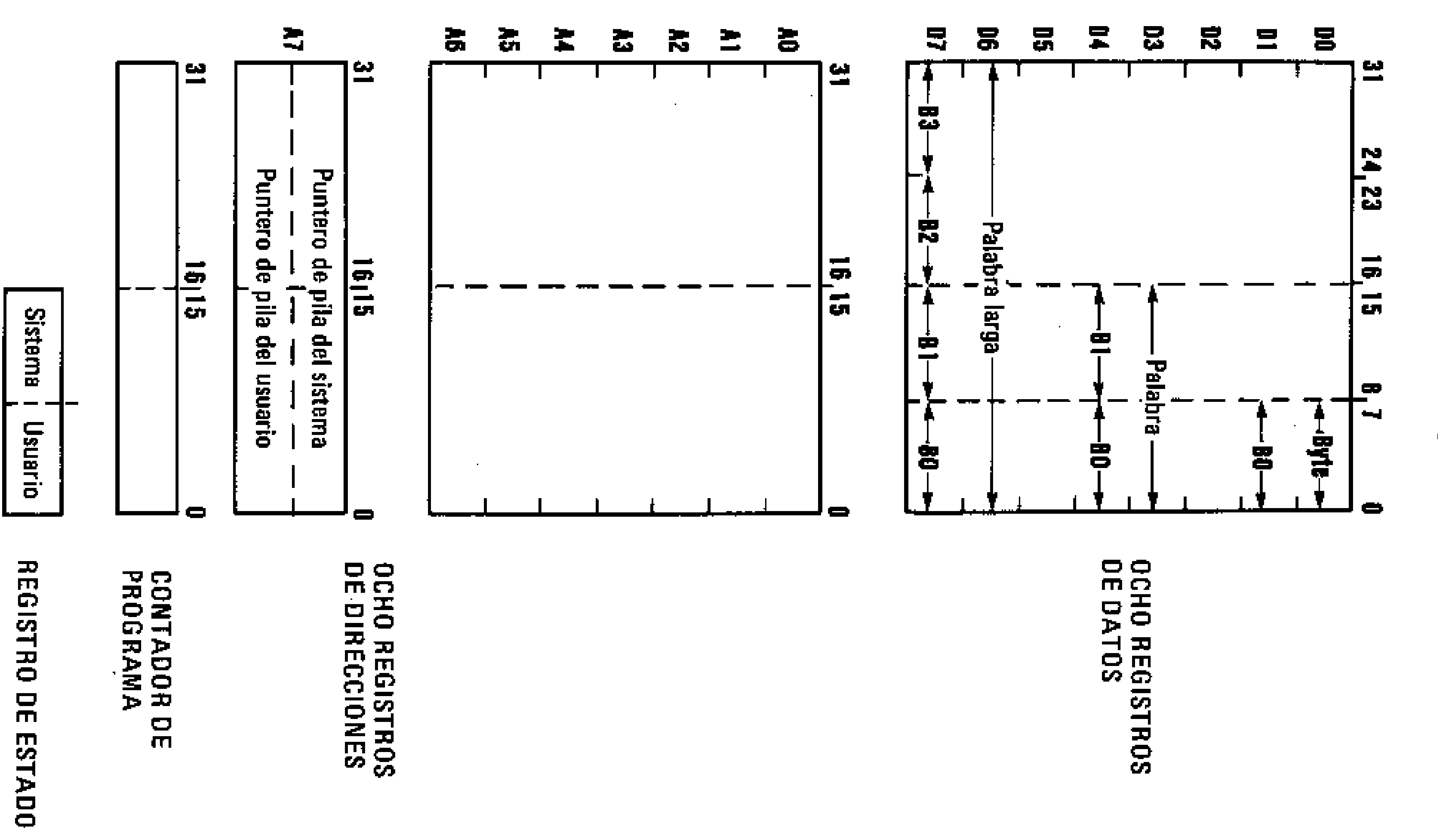


Fig. B.4.– Modelos de programación para el 68008 de Motorola.

Nemónico	Sintaxis del Assembler	Tamaño del operando	Modos de direccionamiento permitidos		Códigos de condición
			Fuente	Destino	X N Z V C
ABCD	ABCD Dy,Dx ABCD −(Ay),−(Ax)	8 8	Dn −(An)	Dn −(An)	* U * U * * U * U *
ADD	ADD <ea>,Dn ADD Dn,<ea>	8, 16, 32 8, 16, 32	All (1) Dn	Dn Alterable	* * * * * * * * * *
ADDA	ADD <ea>,An	16, 32	All	An	− − − − −
ADDI	ADDI #d,<ea>	8, 16, 32	#d	Data Alterable	* * * * *
ADDQ	ADDQ #d,<ea>	8, 16, 32	#d (2)	Alterable (1)	* * * * *
ADDX	ADDX Dy,Dx ADDX −(Ay),−(Ax)	8, 16, 32 8, 16, 32	Dn −(An)	Dn −(An)	* * * * * * * * * *
AND	AND <ea>,Dn AND Dn,<ea>	8, 16, 32 8, 16, 32	Data Dn	Dn Alterable	− * * 0 0 − * * 0 0
ANDI	ANDI #d,<ea> ANDI #d,SR (3)	8, 16, 32 8, 16	#d #d	Data Alterable SR	− * * 0 0 * * * * *
ASL	ASL Dx,Dy ASL #d,Dn ASL <ea>	8, 16, 32 8, 16, 32 16	Dn (4) #d (5)	Dn Dn Memory Alterable	* * * * * * * * * * * * * * *
ASR	ASR Dx,Dy ASR #d;Dn ASR <ea>	8, 16, 32 8, 16, 32 16	Dn (4) #d (5)	Dn Dn Memory Alterable	* * * * * * * * * * * * * * *

Fig. B.5.– Juego de instrucciones para el microprocesador 68000 de Motorola.

Nemónico	Sintaxis del Assembler	Tamaño del operando	Modos de direccionamiento permitidos		Códigos de condición
			Fuente	Destino	X N Z V C
Bcc	Bcc <label>	8, 16	If cc, then PC + d → PC		- - - - -
BCHG	BCHG Dn,<ea> BCHG #d,<ea>	8, 32 8, 32	Dn #d	Data Alterable Data Alterable	- - * - - - - * - -
BCLR	BCLR Dn,<ea> BCLR #d,<ea>	8, 32 8, 32	Dn #d	Data Alterable Data Alterable	- - * - - - - * - -
BRA	BRA <label>	8, 16	PC + d → PC		- - - - -
BSET	BSET Dn,<ea> BSET #d,<ea>	8, 32 8, 32	Dn #d	Data Alterable Data Alterable	- - * - - - - * - -
BSR	BSR <label>	8, 16	PC → -(SP); PC + d → PC		- - - - -
BTST	BTST Dn,<ea> BTST #d,<ea>	8, 32 8, 32	Dn #d	Data, Except Immediate Data, Except Immediate	- - * - - - - * - -
CHK	CHK <ea>,Dn	16	If Dn < 0 or Dn > (ea), then TRAP	Data	- * U U U
CLR	CLR <ea>	8, 16, 32	Data Alterable		- 0 1 0 0
CMP	CMP <ea>,Dn	8, 16, 32	All (1)	Dn	- * * * *
CMPA	CMPA <ea>,An	16, 32	All	An	- * * * *
CMPI	CMPI #d,<ea>	8, 16, 32	#d	Data Alterable	- * * * *
CMPM	CMPM (Ay)+,(Ax)+	8, 16, 32	(An)+	(An)+	- * * * *

Fig. B.5. (cont.)

Nemónico	Sintaxis del Assembler	Tamaño del operando	Modos de direccionamiento permitidos		Códigos de condición
			Fuente	Destino	X N Z V C
DBcc	BDcc Dn,<label>	16	If cc, then Dn - 1 → Dn; if Dn ≠ - 1, then PC + d → PC		- - - - -
DIVS	DIVS <ea>,Dn	16	Data	Dn	- * * * 0
DIVU	DIVU <ea>,Dn	16	Data	Dn	- * * * 0
EOR	EOR Dn,<ea>	8, 16, 32	Dn	Data Alterable	- * * 0 0
EORI	EORI #d,<ea> EORI #d,SR (3)	8, 16, 32 8, 16	#d #d	Data Alterable SR	- * * 0 0 * * * * *
EXG	EXG Rx,Ry	32	Dn or An	Dn or An	- - - - -
EXT	EXT Dn	16, 32	Dn		- * * 0 0
JMP	JMP <ea>		<ea> → PC	Control	- - - - -
JSR	JSR <ea>		PC → -(SP); <ea> → PC	Control	- - - - -
LEA	LEA <ea>,An	32	Control	An	- - - - -
LINK	LINK An,#d	Unsized	An		- - - - -
LSL	LSL Dx,Dy LSL #d,Dn LSL <ea>	8, 16, 32 8, 16, 32 16	Dn (4) #d (5)	Dn Dn Memory Alterable	* * * 0 * * * * 0 * * * * 0 *
LSR	LSR Dx,Dy LSR #d,Dn LSR <ea>	8, 16, 32 8, 16, 32 16	Dn (4) #d (5)	Dn Dn Memory Alterable	* 0 * 0 * * 0 * 0 * * 0 * 0 *

Fig. B.5. (cont.)

Nemónico	Sintaxis del Assembler	Tamaño del operando	Modos de direccionamiento permitidos		Códigos de condición
			Fuente	Destino	X N Z V C
MOVE	MOVE <ea>,<ea> MOVE <ea>,CCR MOVE <ea>,SR (6) MOVE SR,<ea> MOVE USP,An (6) MOVE An,USP (6)	8, 16, 32 16 16 16 32 32	All (1) Data Data SR USP An	Data Alterable CCR SR Data Alterable An USP	- * * 0 0 * * * * * * * * * * - - - - - - - - - - - - - - -
MOVEA	MOVEA <ea>,An	16, 32	All	An	- - - - -
MOVEM	MOVEM <list>,<ea> MOVEM <ea>,<list>	16, 32 16, 32	 Control or (An)+	Control Alterable or –(An) 	- - - - - - - - - -
MOVEP	MOVEP Dx,d(Ay) MOVEP d(Ay),Dx	16, 32 16, 32	Dn d(An)	d(An) Dn	- - - - - - - - - -
MOVEQ	MOVEQ #d,Dn	32	#d (7)	Dn	- * * 0 0
MULS	MULS <ea>,Dn	16	Data	Dn	- * * 0 0
MULU	MULU <ea>,Dn	16	Data	Dn	- * * 0 0
NBCD	NBCD <ea>	8		Data Alterable	* U * U *
NEG	NEG <ea>	8, 16, 32	Data Alterable		* * * * *
NEGX	NEGX <ea>	8, 16, 32	Data Alterable		* * * * *
NOP	NOP		PC + 2 - PC		- - - - -
NOT	NOT <ea>	8, 16, 32		Data Alterable	- * * 0 0

Fig. B.5. (cont.)

Nemónico	Sintaxis del Assembler	Tamaño del operando	Modos de direccionamiento permitidos		Códigos de condición
			Fuente	Destino	X N Z V C
OR	OR <ea>, Dn OR Dn,<ea>	8, 16, 32 8, 16, 32	Data Dn	Dn Alterable	- * * 0 0 - * * 0 0
ORI	ORI #d,<ea> ORI #d,SR (3)	8, 16, 32 8, 16	#d #d	Data Alterable SR	- * * 0 0 * * * * *
PEA	PEA <ea>	32	Control		- - - - -
RESET (6)	RESET				- - - - -
ROL	ROL Dx,Dy ROL #d,Dn ROL <ea>	8, 16, 32 8, 16, 32 16	Dn (4) #d (5)	Dn Dn Memory Alterable	- * * 0 * - * * 0 * - * * 0 *
ROR	ROR Dx,Dy ROR #d,Dn ROR <ea>	8, 16, 32 8, 16, 32 16	Dn (4) #d (5)	Dn Dn Memory Alterable	- * * 0 * - * * 0 * - * * 0 *
ROXL	ROXL Dx,Dy ROXL #d,Dn ROXL <ea>	8, 16, 32 8, 16, 32 16	Dn (4) #d (5)	Dn Dn Memory Alterable	* * * 0 * * * * 0 * * * * 0 *
ROXR	ROXR Dx,Dy ROXR #d,Dn ROXR <ea>	8, 16, 32 8, 16, 32 16	Dn (4) #d (5)	Dn Dn Memory Alterable	* * * 0 * * * * 0 * * * * 0 *

Fig. B.5. (cont.)

Nemónico	Sintaxis del Assembler	Tamaño del operando	Modos de direccionamiento permitidos: Fuente	Destino	Códigos de condición: X N Z V C
RTE (6)	RTE		(SP) + → SP; (SP) + → PC		* * * * *
RTR	RTR		(SP)+ → CCR; (SP) + → PC		* * * * *
RTS	RTS		(SP) + → PC		- - - - -
SBCD	SBCD Dy,Dx SBCD −(Ay),−(Ax)	8 8	Dn −(An)	Dn −(An)	* U * U * * U * U *
Scc	Scc <ea>	8	If cc, then 1s → (ea); otherwise 0s → (ea)	Data Alterable	- - - - -
STOP (6)	STOP #d	16	#d → SR, then STOP		* * * * *
SUB	SUB <ea>,Dn SUB Dn,<ea>	8, 16, 32 8, 16, 32	All (1) Dn	Dn Alterable	* * * * * * * * * *
SUBA	SUBA <ea>,An	16, 32	All	An	- - - - -
SUBI	SUBI #d,<ea>	8, 16, 32	#d	Data Alterable	* * * * *
SUBQ	SUBQ #d,<ea>	8, 16, 32	#d (2)	Alterable (1)	* * * * *
SUBX	SUBX Dy,Dx SUBX −(Ay),−(Ax)	8, 16, 32 8, 16, 32	Dn −(An)	Dn −(An)	* * * * * * * * * *
SWAP	SWAP Dn	16	Dn		- - - - -
TAS	TAS <ea>	8	Data Alterable		- * * 0 0

Fig. B.5. (cont.)

Nemónico	Sintaxis del Assembler	Tamaño del operando	Modos de direccionamiento permitidos: Fuente	Destino	Códigos de condición: X N Z V C
TRAP	TRAP #<vector>		PC → −(SP); SR → −(SP); #<vector> → PC		- - - - -
TRAPV	TRAPV		If V = 1, then TRAP		- - - - -
TST	TST <ea>	8, 16, 32	Data Alterable		- * * 0 0
UNLK	UNLK An	Unsized		An	- - - - -

Footnotes:
(1) If the operation size is byte, the address register direct addressing mode is not allowed.
(2) Immediate operand, with a value from 1 to 8.
(3) If the operation size is word, the instruction is privileged.
(4) Source data register contains the shift count. Count = 0 to 63, where 0 produces a count of 64.
(5) The data is the shift count, 1 to 8.
(6) This operation is privileged.
(7) Eight bits of immediate data, which are sign-extended to a 32-bit long operand.

Fig. B.5. (cont.)

Suffix "cc"	Condition	True If
EQ	igual a	$Z = 1$
NE	no igual a	$Z = 0$
MI	menos	$N = 1$
PL	más	$N = 0$
*GT	mayor que	$Z \wedge (N \veebar V) = 0$
*LT	menor que	$N \veebar V = 1$
*GE	mayor o igual que	$N \veebar V = 0$
*LE	menor o igual que	$Z \vee (N \veebar V) = 1$
HI	mayor que	$C \wedge Z = 0$
LS	menor o el mismo que	$C \vee Z = 1$
CS	Carry set	$C = 1$
CC	Carry clear	$C = 0$
*VS	Overflow	$V = 1$
*VC	No overflow	$V = 0$
T	siempre true	
F	siempre false	

Fig. B.5. (cont.)

APENDICE C

Las palabras reservadas

Las palabras reservadas o palabras clave en SuperBASIC cumplen el objetivo de indicar qué instrucciones deben ser ejecutadas y sólo deben ser escritas allí donde son necesarias dentro del contexto específico del programa.

Estas palabras reservadas se forman siguiendo los criterios de escritura de los identificadores de SuperBASIC, aunque en algunos casos solamente es necesario escribir ciertas letras de las palabras reservadas para que sean significativas dentro de su contexto.

El conjunto que se muestra seguidamente puede ser aumentado con la adición de procedimientos definidos por el usuario y que sean cargados como parte perteneciente al sistema operativo del QL.

Naturalmente, las palabras clave no pueden ser utilizadas como identificadores en programas SuperBASIC.

Palabra clave	Descripción
ABS	Toma el valor absoluto de un número
ACOS	Calcula el arcoseno de un ángulo
ACOT	Calcula el arcotangente de un ángulo
ADATE	Ajusta el reloj un número de segundos
ARC	Dibuja una curva
ASIN	Calcula el coseno de un ángulo
AT	Mueve el cursor de texto a la posición indicada
ATAN	Calcula el arcotangente de un ángulo
AUTO	Genera automáticamente números de línea
BAUD	Fija el rango de velocidad de transferencia a puertas RS232
BEEP	Produce sonido
BEEPING	Detecta el estado de sonido/silencio
BLOCK	Dibuja un rectángulo
BORDER	Fija el borde de una ventana
CALL	Hace ejecutarse una subrutina en código máquina
CHR$	Convierte números en sus correspondientes caracteres ASCII
CIRCLE	Dibuja una circunferencia
CLEAR	Limpia el valor de todas las variables
CLOSE	Cierra la comunicación con un dispositivo
CLS	Limpia una ventana o una sección
CODE	Convierte caracteres ASCII
CONTINUE	Reanuda la ejecución de un programa
COPY	Transfiere datos entre dispositivos
COS	Calcula el coseno de un ángulo
COT	Calcula la cotangente de un ángulo
CSIZE	Fija el tamaño de los caracteres
CURSOR.	Mueve el cursor en la pantalla
DATA	Señala el comienzo de los datos dentro de un prog.
DATE	Almacena la fecha en segundos
DATE$	Almacena la fecha en formato ISO
DAY$	Almacena el día de la semana actual

DEF FN	Declaración de comienzo de una función
DEF PROC	Declaración de comienzo de un procedimiento
DEG	Convierte radianes en grados
DELETE	Borra un fichero de un microdrive
DLINE	Borra líneas de un programa
DIM	Reserva espacio para una matriz
DIMN	Devuelve el tamaño de una matriz
DIR	Visualiza el directorio de un microdrive
DIV	División entera
EDIT	Edición de líneas de programas
ELSE	Se usa con la IF ... THEN
END DEF	Señala el final de una definición
END FOR	Señala el final de un bucle FOR
END REPEAT	Señala el final de un bucle REPEAT
END SELECT	Señala el final de una instrucción SELECT
EXEC	Llama a un programa en código máquina desde un microdrive
EXIT	Permite la salida de una estructura de bucle
EXP	Eleve *e* a una potencia dada
FILL	Rellena un dibujo
FILL$	Repite una cadena de caracteres
FLASH	Produce el parpadeo de un texto
FOR	Señala el comienzo de un bucle FOR
FORMAT	Inicializa un microdrive
GOSUB	Llama a una subrutina
GOTO	Salta a la línea especificada
IF	Verifica una condición
INK	Fija un color de trabajo
INKEY$	Explora un dispositivo para dar entrada a un carácter
INPUT	Recibe datos desde un dispositivo
INSTR	Fracciona una cadena
INT	Redondea números reales

KEYROW	Explora la matriz del teclado para una entrada
LBYTES	Carga bytes en un área de memoria
LEN	Devuelve la longitud de una cadena
LINE	Dibuja una recta
LIST	Visualiza el programa en memoria
LN	Devuelve el logaritmo natural (base e)
LOAD	Carga un programa desde un dispositivo
LOG10	Calcula el logaritmo común (base 10)
LRUN	Carga y ejecuta un programa
MERGE	Fusiona varios ficheros
MOD	Devuelve el resto de una división
MODE	Define el modo gráfico de la pantalla
MOVE	Mueve la tortuga en una dirección
MRUN	Fusiona y ejecuta un programa
NET	Define un número de estación
NEW	Limpia el área de programas
NEXT	Continuación de un bucle FOR ... NEXT
ON GOTO	Verificación de una variable de bifurcación
OPEN	Abre un fichero antes de su uso
OVER	Controla la forma de impresión de un texto
PAN	Desplaza la pantalla horizontalmente
PAPER	Define el color de fondo
PAUSE	Espera un período de tiempo
PEEK	Examina el contenido de un segmento de memoria
PENUP	Levanta el lápiz del papel
PENDOWN	Baja el lápiz al papel
PI	Equivalente a 3.141593
POINT	Dibuja un punto
POKE	Modifica el contenido de un segmento de memoria
PRINT	Visualiza textos en algún canal

RAD	Convierte grados en radianes
RANDOMISE	Prepara el generador de números aleatorios
READ	Los valores de sentencias DATA
RECOL	Altera los colores de la pantalla
REMARK	Comentarios
RENUM	Renumera las líneas de un programa
REPEAT	Señala el comienzo de un bucle REPEAT
RESTORE	Restaura las sentencias DATA
RETRY	Continúa después de una interrupción
RETURN	Retorno desde una subrutina
RND	Proporciona un número aleatorio
RUN	Comienza la ejecución de un programa
SAVE	Salva un programa
SBYTES	Salva un segmento de memoria
SCALE	Ajusta la escala gráfica
SCROLL	Desplaza la pantalla verticalmente
SDATE	Fija el reloj del sistema y la fecha
SELECT	Selecciona una variable de control
SEXEC	Salva una rutina multitarea
SIN	Calcula el seno de un ángulo
SQRT	Calcula la raíz cuadrada
STOP	Provoca la parada de un programa
STRIP	Altera el color del patrón
TAN	Devuelve la tangente de un ángulo
THEN	En conjunción con IF...THEN ... ELSE
TURN	Gira la tortuga un ángulo determinado
TURNTO	Gira la tortuga a un ángulo determinado
UNDER	Controla el subrayado
WINDOW	Crea una ventana

Bibliografía

Erskine, R.: *Pocket Guide: Assembly Language Programming for the MC68000 Series Processors.* Pitman, 1984.

Kane, G.: *68000 Microprocessor Handbook.* OSBORNE/Mc Graw-Hill, 1978

King, T. y Knight, B.: *Programming the MC68000.* Adison and Wesley, 1983

Scanlon, L. J.: *The 68000. Principles and Programming.* Sams, 1981

Titus, C.A.: *16 Bit Microprocessors.* Sams 1981

Angulo, J.M.: *Microprocesadores de 16 bits.* Paraninfo, 1984

Mosket: *MK 68000 16-bit Microproccessor.*

Motorola: *Users manual MC68000.* Prentice Hall

Jaulent: *Le microprocesseur 68000 et sa programation.* Eyrolles.

Dickens, A.: *QL Advanced User Guide.* Adder, 1984

Motorola: *MC68000 Cross Macro Assembler Reference Manual,* 1979

Indice de conceptos

A

B

C

D

E

F

G

H

I

J

K

L

WATT y MANGADA.– BASIC para niños. 5ª edición.
WATT y MANGADA.– BASIC avanzado para niños. 3ª edición.
WATT y MANGADA.– BASIC para niños con el microordenador DRAGON.

Aplicaciones e Informática Profesional

ANGULO.– Curso de Robótica. 2ª edición.
ANGULO.– Robótica práctica. Teoría y aplicaciones.
ANGULO.– Prácticas de Microelectrónica y microinformática. 2ª edición.
ASPINALL.– El microprocesador y sus aplicaciones.
BANKS.– Microordenadores. Cómo funcionan. Para qué sirven.
BELLIDO.– Amaestra tu DRAGON. Curso de programación en BASIC para el microordenador DRAGON.
BELLIDO.– ZX81. Curso de programación en BASIC. 3ª edición.
BELLIDO.– Cómo programar su Spectrum y Timex 2068. 7ª edición.
BELLIDO.– Cómo usar los colores y los gráficos en el Spectrum. 3ª edición. (Libro y casete).
BELLIDO.– KIT de gráficos para Spectrum.
BELLIDO.– Enciclopedia del Spectrum. Tomo 1.
BELLIDO.– Spectrum. Iniciación al Código Máquina.
ELLERSHAW y SCHOFIELD.– Las primeras 15 horas con el Spectrum. 2ª edición.
ERSKINE.– Los mejores programas para el ZX Spectrum.
ESCUDERO.– (Centro de Investigación UAM-IBM). Reconocimiento de patrones. Fundamentos teóricos, algorítmos y aplicaciones de la moderna técnica denominada "Pattern Recognition".
FERRER.– Programas en BASIC.
GAUTHIER y PONTO.– Diseño de programas para sistemas. 4ª edición.
HARTMAN, MATTHES y PROEME.– Manual de los sistemas de información. 2 tomos. 7ª edición.
LEPAPE.– Programación del Z80 con ensamblador.
LUCAS, JR.– Sistemas de información. Análisis. Diseño. Puesta a punto.
MARTINEZ VELARDE.– El libro de Código Máquina del Spectrum. 2ª edición.
MONTEIL.– Cómo programar su Commodore 64. 2 tomos. Tomo 1: BASIC. Gráficos. Sonidos. 4ª edición. Tomo 2: Lenguaje máquina. Entradas-salidas y periféricos.
PANNELL, JACKSON y LUCAS.– El microordenador en la pequeña Empresa.
PLOUIN.– IBM-PC. Características. Programación. Manejo.
QUANEAUX.– Tratamiento de textos en BASIC.
WILLIAMS.– Programación paso a paso con el Spectrum. 2ª edición.

PROGRAMACION Y PRACTICA CON EL SINCLAIR QL

Este libro está concebido fundamentalmente como una obra didáctica destinada a todos aquellos que posean o utilicen el Sinclair QL y aún más, a todos aquellos que deseen introducirse en el terreno de la programación estructurada utilizando un lenguaje de alto nivel ya bastante evolucionado como es el SuperBASIC.

La información contenida es válida tanto para los no iniciados, sin conocimientos de informática, como para los que ya poseen estas enseñanzas de BASIC y de microordenadores.



Magallanes, 25.—Madrid-15.

ISBN: 84-283-1424-1